Jin Xu

Biological Computing

OPEN ACCESS

Springer

Biological Computing

Jin Xu

Biological Computing

Springer

Jin Xu
School of Computer Science
Peking University
Beijing, China

ISBN 978-981-96-3869-7 ISBN 978-981-96-3870-3 (eBook)
https://doi.org/10.1007/978-981-96-3870-3

This work was supported by Jin Xu.

The original submitted manuscript has been translated into English. The translation was done using artificial intelligence. A subsequent revision was performed by the author(s) to further refine the work and to ensure that the translation is appropriate concerning content and scientific correctness. It may, however, read stylistically different from a conventional translation.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd.
The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

This book is dedicated to Professor Ke Liu, with sincere appreciation for his contributions to the advancement of biological computing both in China and globally.

Preface

In 1996, I was deeply engaged in addressing complex graph theory challenges, such as the Traveling Salesman Problem, the Graph Coloring Problem, and the Hamiltonian Problem, utilizing Artificial Neural Networks (ANNs). These problems are categorized as NP-complete, and I aspired to develop efficient algorithms for them with the help of ANNs. I guided my doctoral students in publishing several academic papers focused on tackling NP-complete problems through the use of ANNs. One day, one of my doctoral students mentioned discovering an article in *Science* that presented a method for solving the Directed Hamiltonian Path Problem using DNA molecules. I was exhilarated by this finding, convinced it could offer a novel approach to addressing NP-complete problems, and I was eager to delve deeper into the subject. I rushed to the library to photocopy the article and began reading it. However, as I immersed myself in the text, I felt a sense of confusion—having never studied biology in high school, I struggled to understand the biological concepts and experimental designs described in the paper.

From that moment on, I resolved to make a significant contribution to the field of biological computing by learning biology. I started from the ground up, initially focusing on high school-level biology before progressing to more advanced topics in molecular biology and biochemistry. After over a decade of dedicated effort, I became proficient in independently executing essential stages of utilizing DNA computing to address NP-complete problems, including mathematical model construction, experimental design, and solution verification.

Over the past 30 years, DNA computing has progressed from enumeration-based models to non-enumerative and parallel models. A significant milestone occurred between 2002 and 2016 with the introduction of the probe machine, which represented a groundbreaking advancement in the field. The probe machine is a fully parallel mathematical computing model developed to establish a DNA model for graph coloring. Its advent has provided computer science with a new mathematical computing framework that distinguishes itself from the Turing machine, garnering considerable attention from scholars and industry professionals alike.

This book provides a comprehensive overview of the research accomplishments of scholars worldwide in the realm of biological computing over several decades.

The main topics encompassed include fundamental theories, key technologies, experimental methodologies, computational models, and application examples of DNA computing. In the section dedicated to foundational theory, we detail the structural characteristics of DNA molecules, the mechanisms of biological enzymes, and the fundamental principles underlying DNA computing. This content lays the groundwork for readers to comprehend the operation of DNA computing. The chapters focusing on key technologies and experimental methodologies examine essential techniques such as DNA encoding, PCR technology, and gel electrophoresis. These techniques are critical for DNA computing, enabling readers to understand the vital aspects of experimental design and execution. In our exploration of computational models, we delve into various DNA computing frameworks, including enumeration-based, non-enumerative, and parallel models. We discuss the construction concepts behind these models, along with their respective advantages and disadvantages, providing diverse options for addressing different types of computational challenges. Furthermore, we present several case studies showcasing the applications of DNA computing in fields such as cryptography, bioinformatics, and optimization problems, highlighting the extensive potential of this innovative computing approach.

The strength of this book lies in its systematic, innovative, and practical approach. The systematic nature is evident in the comprehensive organization and integration of knowledge pertaining to DNA computing, further enhanced by an in-depth introduction to RNA computing and protein computing. By covering everything from fundamental concepts to applications and effectively bridging theory with practice, it establishes a robust knowledge framework. The innovative aspect is showcased through a timely exploration of the latest advancements in DNA computing research, including a thorough analysis of pioneering technologies such as probe machines and DNA self-assembly, ensuring that readers are well-informed about current developments in the field. The practical component is emphasized through detailed descriptions of experimental procedures and clear explanations of methods for constructing computational models, offering valuable guidance for those involved in related research.

In our writing methodology, we have integrated both theoretical and practical approaches. Initially, we undertook comprehensive research and systematic summarization of the fundamental theories and essential technologies related to DNA computing by reviewing a substantial body of literature. Building on our practical experience gained from DNA computing experiments, we then provided a detailed elaboration of the experimental procedures and key precautions to consider.

This book is designed for researchers in areas such as graph theory and algorithms, bioinformatics, computer science, and molecular biology, as well as for university students and self-learners with an interest in biological computing. For researchers, it serves as a valuable reference on DNA computing, providing insights into the latest developments and technological advancements in the field, along with innovative ideas and methodologies for their scientific endeavors. For university students, this book is appropriate as both a textbook and a reference for biological computing courses, enabling them to systematically acquire essential

knowledge and skills in DNA computing, thereby establishing a firm foundation for their future studies and research. For self-learners, the book's clear language and abundant examples facilitate a swift understanding of the core concepts of DNA computing, enhancing their overall comprehension of the subject.

When engaging with this book, readers should consider the following points: Firstly, as DNA computing encompasses various disciplines, a foundational understanding of biology, computer science, and graph theory is essential for a more comprehensive grasp of the material. Secondly, the experimental procedures and computational model-building techniques presented in this book call for practical exploration and validation. Therefore, it is advisable to integrate hands-on activities with the reading to deepen understanding and enhance practical skills. Furthermore, considering the rapid advancements in DNA computing, readers should keep an open mind and stay informed about the latest research developments and technological trends to continuously enhance their knowledge.

This book has greatly benefited from the invaluable support and assistance of Enqiang Zhu, Xiaoli Qiang, Gang Fang, Zheng Kou, Xiaolong Shi, Wenbin Liu, Yan Wang, Zhihua Chen, Zehui Shao, Fengyue Zhang, Kai Zhang, and Congzhou Chen, among others. The manuscript, particularly the illustrations, has undergone more than a year of rigorous revision and refinement, reflecting their dedicated efforts. I would like to express my heartfelt gratitude to each of them! Additionally, Chanjian Liu, Xiaoqing Liu, and Huang Leng have meticulously proofread the entire book, and I am equally thankful for their contributions.

Due to time constraints, there may inevitably be some shortcomings and errors in the book. I appreciate any criticism and suggestions from readers.

We sincerely hope that this book will offer valuable knowledge and inspiration, as well as contribute to the ongoing development of research and applications in DNA computing. I also look forward to engaging in in-depth discussions with readers and colleagues, collectively fostering the advancement of biological computing.

Beijing, China Jin Xu
February 7, 2025

Declarations

Competing Interests The author has no competing interests to declare that are relevant to the content of this manuscript.

Contents

Chapter 1
Introduction

Biological computation refers to the computation that uses biological macromolecules as data for information processing. Biological macromolecules mainly include DNA, RNA, and proteins, and consequently, biological computation can be divided into DNA computation, RNA computation, and protein computation. Limited by the level of biochemical operation technology, biological computation research is currently mainly focused on DNA computation. This book focuses on DNA computation and also gives a certain introduction to RNA computation and protein computation. This chapter introduces the background of the emergence of biological computation, research significance, and research progress.

1.1 Background of the Emergence of Biological Computation

The advancement of human civilization is intricately connected to the evolution of computational tools designed for information processing. These tools reflect the level of human development and act as primary drivers of societal progress. Throughout history, humanity has traversed several distinct eras: the Stone Age, Iron Age, Steam Age, Electrical Age, Information Age, and now the era of Artificial Intelligence. In these epochs, computational tools have progressed from simple to complex and from basic to sophisticated forms. This journey began with methods like knotting records and using counting rods, evolving through devices such as the abacus, slide rules, and mechanical computers, ultimately leading to electronic computers. The electronic computer itself has transformed from vacuum tubes and transistors to personal computers (PCs) and today's supercomputers. Each stage of this development has played a crucial role in advancing human society throughout various historical periods. Figure 1.1 provides a visual representation of the evolution of computational tools.

Indeed, the human biological nervous system has consistently been the most effective tool for processing information. This system has evolved over time, result-

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_1

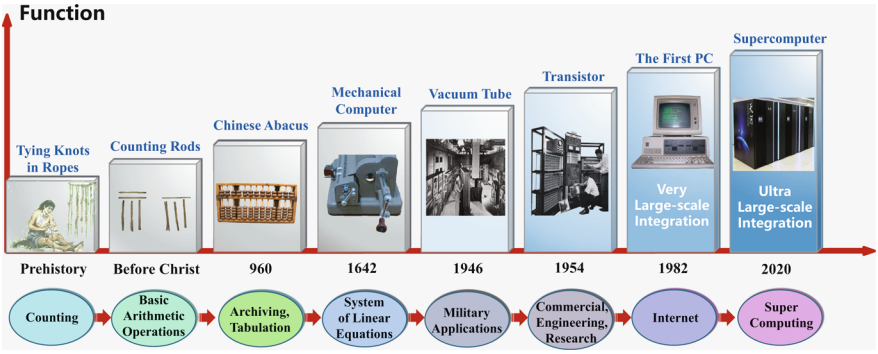


Fig. 1.1 The development process of human computational tools

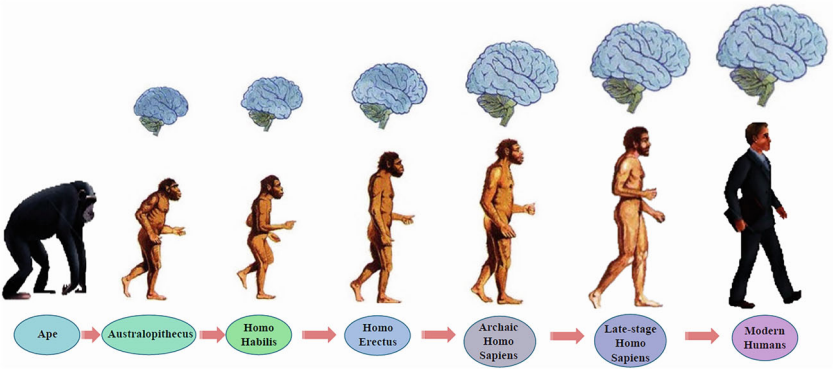


Fig. 1.2 The evolution of the human nervous system

ing in a progressively more complex brain. Figure 1.2 presents a schematic diagram that illustrates the evolution of the human nervous system. The human brain, as an information-processing system, not only evolves itself but also designs and creates all other human information-processing tools. Among these, the electronic computer emerges as the most advanced computational device developed by humanity.

In today's society, the electronic computer is the core tool for information processing, deeply intertwined with every aspect of human life. However, the computational model employed by electronic computers is based on the Turing machine, which functions through serial computation. This limitation means that electronic computers are not well-equipped to tackle **NP**-complete problems effectively. These problems are characterized by a computational demand that increases exponentially with the problem's size. Furthermore, as the manufacturing technology for electronic computers has reached its limits, this has prompted a search for new computational models. The aim is to develop innovative computational tools capable of overcoming the challenges posed by **NP**-complete problems, which serve as obstacles to societal advancement.

In recent decades, scientists have delved into various fields to pursue innovative computational models. They have introduced several approaches, including artificial neural networks that emulate the brain's information processing, evolutionary computation that reflects genetic mechanisms, biological computation grounded in the properties of biological macromolecules, quantum computation harnessing quantum characteristics, and optical computation utilizing the properties of light. As these computational models advance and reach a certain level of maturity, they often lead to the development of corresponding computing systems. Therefore, it is appropriate to compare the current landscape of non-traditional computing to a "Warring States Period," with biological computation serving as a key player in this evolving arena.

Biological computation primarily focuses on addressing NP-complete problems. Accordingly, Chap. 2 provides a comprehensive overview of fundamental concepts, including basic notations and terminologies in graph theory, the Turing machine model, and computational complexity theory as grounded in the Turing machine framework.

The origins of biological computation can be traced back to Richard P. Feynman [1], who proposed the concept of a sub-micron scale computer in 1961. In 1973, Charles H. Bennett [2] introduced the idea of a Turing machine that operates through enzymatic catalysis. In the 1980s, researchers developed a binary molecular computing model that employed proteins for information processing, a field known as protein computing. This model was inspired by the advancements in electronic computers, with the goal of using molecules to create controllable "binary" states that represent the binary values of "0" and "1." A comprehensive discussion on protein computing will be provided in Chap. 12.

In 1994, Leonard Adleman [3] introduced a DNA computing model to tackle the Hamiltonian path problem in a directed graph with seven vertices. This model operates on the principle of using DNA molecules as "data" while employing biological enzymes and PCR instruments as "operators" to derive the solution to the problem, as elaborated in Chap. 6. Additionally, a significant factor in the emergence of the DNA computing model was the human genome sequencing project. The execution of this extensive project has fueled rapid advancements in genetic engineering, molecular biology theory, and biochemical operation technology, especially in DNA sequencing technology. This progress has cultivated a fertile environment for the emergence, realization, and development of DNA computing, establishing a strong foundation for research and development in DNA computers.

This book explores the significant achievements in DNA computing research, covering essential topics such as fundamental theory, experimental operations, and computing models, all of which will be elaborated upon in Chaps. 3, 4, 5, 6, 7, and 8. References [4–8] include several early review articles by the author in this field. Throughout the research on DNA computing, a fully parallel computing model known as the probe machine [9] will be examined in Chap. 9, including its theoretical framework and practical applications. Chapter 10 will delve into DNA molecule self-assembly technology, which has emerged from DNA computing

investigations. Lastly, Chap. 11 will address recent advancements in the RNA computing model.

1.2 General Definition and Classification of Computers

The definition of an electronic computer does not adequately encompass the broader concept of a computer. This section provides a formal definition of a computer and offers a classification based on this definition.

In straightforward terms, a computer is a machine designed according to a computing model that demonstrates a certain level of universality. This can involve either identifying materials that are compatible with this computing model or developing a computing model based on specific types of materials. Once the appropriate materials and computing model are determined, the development process necessitates the design of a system structure, commonly referred to as the architecture. With this in mind, we can formally define a computer as follows:

Computer = Computing Model + Architecture + Implementation Material.

For instance, the underlying computing model of an electronic computer is the Turing machine. The materials used for implementation consist of electronic components such as diodes, triodes, resistors, and capacitors. The architecture adheres to the von Neumann model, which comprises five key components. The first part is the arithmetic unit, which is responsible for executing various arithmetic and logical operations and facilitating data transfer essential for data processing. The second component is the controller, which oversees the execution of programs and, in conjunction with the arithmetic unit, forms the Central Processing Unit (CPU) of the computer. The third component is the memory, which stores programs and data. The fourth component is the input devices, including the mouse and keyboard, which are used to enter data or programs into the computer. The last component is the output devices, such as monitors and printers, which display the results of data processing and program execution to the user. These five fundamental components work together under the control of instructions, enabling data transfer among them.

John von Neumann designed the electronic computer, but the most esteemed award in the field of computers today is the "Turing Award," not the "von Neumann Award." This distinction emphasizes the significance of the computing model within modern computers.

Presently, computers and computational models are commonly named based on the materials they employ or the scale of materials. For example, we refer to electronic computers and biological computers (which include DNA, RNA, and protein computers) according to their respective materials, while quantum computers are named based on the scale of materials. However, the essence or core of a computer truly resides in its computing model, which renders the naming conventions for these computer types somewhat questionable. Nonetheless, it is worth noting that whether in practical applications or experimental contexts, all of these computers utilize Turing machines as their foundational computing model.

From this standpoint, it is reasonable to categorize computers by the materials or scales they involve.

The previous classification of computers was based on their implementation materials. Now, we categorize computers into four main types based on computing models.

(1) Serial computing models: These models process information one step at a time. A prime example is the Turing machine, which represents a standard serial computing model. Many early mechanical computers operated using this model.
(2) Fully parallel computing models: An illustration of this type is the probe machine discussed in Chap. 9. A key characteristic of fully parallel computing models is their operation in multiple dimensions, typically three-dimensional.
(3) Serial/parallel computing models: An example of this category is the way the human brain processes information. The brain comprises five parallel sensory systems; however, each system functions serially during the information processing stage. This model highlights the complexity and intelligence inherent in serial/parallel processing.
(4) Intelligent computers: This book does not explore intelligent computers in detail, and it provides its formal definition for clarity.
Intelligent Computer = Computing Model + Architecture + Implementation Material + Basic Intelligence Set

1.3 Significance and Research Progress of Biological Computing

Since 1994, DNA computing has built a 30-year history. Throughout this period, it has largely focused on tackling **NP**-complete problems, aiming to advance the development of DNA computers. Moreover, the DNA self-assembly technology that emerged from this research has greatly enhanced the field of DNA nanodevices. These nanodevices show significant potential for nanorobot applications, especially in tumor diagnosis and drug delivery.

The key reason DNA computing distinguishes itself in the field of biological computing is that the manipulation of DNA molecules and associated biochemical technologies is more readily achievable than that of RNA and proteins. Additionally, DNA computing presents four noteworthy advantages (see references [4–8]):

(1) High parallelism. In DNA computing, the “data” is represented by DNA molecules, while the “operations” involve fundamental processes such as unwinding DNA double strands and connecting or cutting single strands. These operations can occur simultaneously, allowing for significant parallel processing capabilities. The vast quantity of DNA molecules enhances the parallelism of DNA computing even further.

(2) Impressive information storage capacity. The average length of the four bases that comprise DNA strands (adenine, cytosine, guanine, and thymine) is merely 0.34 nm. For example, a DNA strand consisting of 1000 bps[1] measures only 340 nm. It is estimated that 1 cubic meter of DNA can store approximately 10 trillion trillion bits of binary data, which far surpasses the total storage capacity of all electronic computers combined around the world.
(3) Extremely low energy consumption. DNA computing consumes energy at an incredibly low rate. It is roughly estimated that the energy required to solve a problem using DNA computing is only one billionth of the energy that an electronic computer would expend for the same calculation.
(4) Diverse range of operations. DNA computing is characterized by a wide array of operations, including the denaturation of double-stranded DNA, the connection of single-stranded DNAs, the cutting of DNA molecules into fragments, and the replication of DNA strands (polymerase). It also includes techniques from established biochemical methods, such as PCR amplification and electrophoresis.

These compelling advantages of DNA computing have drawn the interest of researchers across various disciplines, including bioengineering, computer science, mathematics, physics, chemistry, control science, and information technology.

The objective of studying DNA computing is to advance the development of practical DNA computers. Current research shows that this field is steadily approaching real-world applications. For example, when searching for all paths between two vertices in a weighted graph, researchers can use DNA strands that correspond to these two vertices as primers and conduct a PCR amplification operation to retrieve all possible paths. Another instance of this research occurred in 2002 when Braich et al. [10] tackled the SAT problem, conducting a search that involved 2^{20} possibilities. Furthermore, reference [11] established a DNA computing model to solve the graph vertex coloring problem, successfully identifying all solutions for a 61-vertex 3-coloring challenge, with the number of searches reaching 3^{59}. This represents the largest search scale achieved in biological computing to date and further illustrates the vast potential of DNA computing.

While there are numerous **NP**-complete problems, in 1971, Stephen Cook established that all **NP**-complete problems can be reduced to one another in polynomial time (for which Cook earned the Turing Award in 1982). This results in the ability to transform any **NP**-complete problem into the graph coloring problem, a typical **NP**-complete problem. Recent advancements in DNA computing have demonstrated significant potential for effectively addressing the graph coloring problem, showcasing the advantages of this methodology for tackling **NP**-complete challenges more broadly. Every progress in solving **NP**-complete problems can lead

[1] A base pair (bp) is a chemical structure formed by hydrogen bonds between the bases on one strand of a nucleic acid molecule and the complementary bases on the other strand in the double helix structure. It is commonly used as a unit to represent the length of DNA or double-stranded RNA. The commonly used units are kilobase pairs (kbp) and megabase pairs (Mbp).

to far-reaching benefits across various sectors of society. For instance, it could result in breakthroughs in areas such as protein structure prediction, train scheduling, and flight path planning. Furthermore, there is optimism that advancements in this domain could help resolve some of the more difficult issues in basic science, particularly in mathematics, operations research, and theoretical computer science.

The research on DNA computing can be categorized into five distinct stages based on the characteristics of the model.

(1) Enumeration stage (1994–2005). This eleven-year period marked the beginning of DNA computing. During this time, researchers were actively exploring various facets of the field. They concentrated on addressing complex **NP**-complete problems, investigating how to develop computational models that utilized the unique properties of DNA molecules. Key areas of focus included coding DNA strands, conducting biochemical experiments, and implementing techniques for solution detection—essentially learning how to extract solutions from the biochemical reaction pool. At this stage, many scholars mistakenly believed that DNA computing could capitalize on the small size of DNA molecules to exchange space for time. An overview of the DNA computing models from this period will be provided in Chap. 6.

(2) Non-enumeration stage. This stage commenced in 2006. It is clear that if the enumeration method were employed to solve the 4-coloring problem for a graph with 200 vertices, the required DNA molecular weight would surpass that of the Earth. Consequently, DNA computing must adopt a non-enumerative approach. In 2006, the author's team introduced the first non-enumeration graph coloring DNA computing model. A detailed discussion of non-enumeration DNA computing models will be provided in Chap. 7.

(3) Parallel DNA computing model. This model was initiated in 2007. The author's team proposed a parallel DNA computing model based on the non-numeration model. The primary innovation of this model is parallelism: it divides a given graph into several subgraphs and determines the coloring for each subgraph. A parallel PCR continuous deletion technique is introduced to eliminate non-solutions during this process. Ultimately, the solutions from all subgraphs are integrated, similar to removing non-solutions across the entire graph. A comprehensive parallel DNA computing model introduction will be presented in Chap. 8.

(4) Probe machine. This concept emerged in 2002. While developing the graph coloring DNA computing model, a novel method was discovered that enables the extraction of all k-colorings from a given graph using a single operation, creating a new type of fully parallel computing model—the probe machine. This model encompasses both biological and electronic implementations, along with various applications, which will be outlined in Chap. 9.

References

1. Feyman, R.P.: There's plenty of room at the bottom. In: Minaturization, D.H. Gilbert (ed.), pp. 282–296. Reinhold, New York (1961)
2. Bennett, C.H.: On constructing a molecular computer. IBM J. Res. Dev. **17**, 525–532 (1973)
3. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(5187), 1021–1024 (1994)
4. Xu, J., Zhang, L.: Principles, Progress, and Challenges of DNA Computing (I): Biological Computing Systems and Their Applications in Graph Theory. J. Comput. Sci. **26**(1), 1–11 (2003)
5. Xu, J., Huang, B.: DNA Computing: Principles, Progress, and Challenges (II) The Formation of Computer "Databases"—Synthesis of DNA Molecules. J. Comput. Sci. **28**(10), 1583–1591 (2005)
6. Xu, J., Zhang, S., Fan, Y., Guo, Y.: Principles, Progress, and Challenges of DNA Computing (III): Data Structures and Properties in Molecular Computing. J. Comput. Sci. **30**(6), 869–880 (2007)
7. Xu, J., Tan, G., Fan, Y., Guo, Y.: DNA Computing: Principles, Progress, and Challenges (IV): On DNA Computing Models. J. Comput. Sci. **30**(6), 881–893 (2007)
8. Xu, J., Li, F.: Principles, Progress, and Challenges of DNA Computing (V): Fixation Technology of DNA Molecules. J. Comput. Sci. **22**, 2283–2299 (2009)
9. Xu, J.: Probe Machine. IEEE Trans. Neural Netw. Learn. Syst. **27**(7), 1405–1416 (2016)
10. Braich, R.S., Chelyapov, N., Johnson, C., et al.: Solution of a 20-variable 3-SAT problem on a DNA computer. Science **296**(5567), 499–502 (2002)
11. Xu, J., Qiang, X., Zhang, K., et al.: A DNA computing model for the graph vertex coloring problem based on a probe graph. Engineering **4**(1), 61–77 (2018)

Chapter 2
Graphs and Computational Complexity

Chapter 1 pointed out that **NP**-complete problems are a "stumbling block" hindering the development of today's technology. Due to the natural advantage of DNA computing's parallelism in solving **NP**-complete problems, research on DNA computing over the past decades has mainly focused on solving NP-complete problems. Considering that many NP-complete problems are graph theory problems, this chapter first introduces some basic knowledge in graph theory; then, it gradually reveals the true nature of NP-complete problems and introduces some related theories of NP-complete problems, especially computational complexity theory.

2.1 Graph Theory Basics

This section offers essential definitions, notations, and theoretical concepts of graph theory that are employed throughout this book.[1] The topics discussed include the definition and types of graphs, the degree sequence, graph operations, graph isomorphism, matrix representations of graphs, and the principles of graph coloring theory.

2.1.1 Definition and Types of Graphs

For a positive integer k, let $X^{(k)}$ denote the collection of all k-element subsets of a non-empty set X. A graph is defined as follows: Given a non-empty set V and its 2-element subset $E \subseteq V^{(2)}$, the ordered pair (V, E) is referred to as a **graph**,

[1] This section is adapted from Chap. 1 of the author's book: *Theory of Maximum Planar Graphs: Structure-Construction-Coloring* [1], with appropriate modifications and enhancements for the specific requirements of the present book.

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_2

denoted by G. In this context, V is identified as the **vertex set**, while E is termed the **edge set** of graph G. The components of V and E are recognized as the **vertices** and **edges** of G, respectively. The notation uv is frequently utilized to signify the edge $\{u, v\} \in E$, with vertices u and v being designated as the **endpoints** of this edge. If $e = uv$ is an edge of G (i.e., $uv \in E$), its two endpoints u and v are described as being **adjacent**. Additionally, vertex u and vertex v are considered **incident** with the edge e, and vice versa. When two edges share a common vertex, they are said to be **adjacent** each other. Given a vertex $v \in V$, its **neighborhood**, denoted $N_G(v)$ (or simply $N(v)$), is the set of all vertices that are adjacent to v in the graph. Let $I \subset V$. If no two vertices in I are adjacent, then I is called an $independent\ set$ of the graph. Likewise, let $M \subset E$. If no two edges in M are adjacent, then M is referred to as a $matching$ or an $edge-independent\ set$ of the graph. For any specified graph G, the vertex set and edge set of G are commonly denoted as $V(G)$ and $E(G)$, respectively.

Remark 2.1 Note that the vertex set V and edge set E in a graph have no repeated elements (as repetition is not permitted in a set) and $V^{(2)}$ is a set composed of unordered 2-element subsets of V. Consequently, there are no repeated edges in E, the two endpoints of each edge in E are different, and the notations uv and vu represent the same edge. Such graphs are called **simple undirected graphs**.

Let $G = (V, E)$ be a graph. If both V and E are finite, then G is called a **finite graph**; otherwise, an **infinite graph**. The graphs mentioned in this book all refer to finite graphs. Usually, the number of vertices in G is called the **order** of G, and the number of edges in G is called the **size** of G. An (n, m)-graph is a graph with order n and size m.

A graph G can be represented as a geometric figure in a plane, with each vertex depicted as a small point and each edge illustrated as a line connecting its two endpoints. This graph depiction is called G's graphical representation. Such representations enhance our understanding of the graph's structure, providing clearer insight into its properties. This capability is one of the compelling aspects of graph theory.

Let $G = (V, E)$ be a n-order simple graph. If $E = V^{(2)}$, then G is called an n-order **complete graph**, denoted as K_n; Fig. 2.1a,b presents two different graphical representations of the 4-order complete graph K_4. The characteristic of a complete graph is that every pair of vertices are adjacent. The exact opposite of a complete graph is: if every pair of vertices in G are not adjacent, that is, $E = \emptyset$, then G is called an n-order **null graph**, denoted as N_n. When not considering the number of vertices, it is usually called an empty graph or a null graph. Let $V = \{v_1, v_2, \ldots, v_n\}$. If $E = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n\}$, then G is called a **path of length** $n-1$ or an $(n-1)$-**path**, denoted as P_{n-1}; see Fig. 2.1c a graphical representation of P_4. If $E = \{v_1v_2, v_2v_3, \ldots, v_{n-1}v_n, v_nv_1\}$, then G is called n-**cycle**, denoted as C_n; see Fig. 2.1d a graphical representation of C_4.

Remark 2.2 In a graphical representation of a graph, the geometric characteristics of vertices and edges are not of primary importance. Specifically, the dimensions of

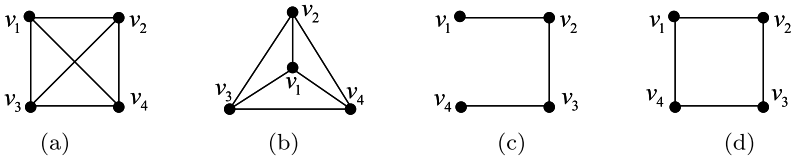


Fig. 2.1 Diagrams of K_4, P_4, and C_4: (**a**) and (**b**) Two graphical representations of K_4; (**c**) a graphical representation of P_4; (**d**) a graphical representation of C_4

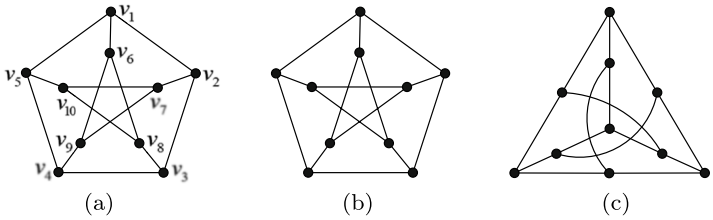


Fig. 2.2 Petersen graph and its different graph representations

the vertices—whether they are round or flat, hollow or solid—and the attributes of the edges, including their length, thickness, and straightness, do not influence the representation.

Example 2.1 Let $G = (V, E)$ be a graph with $V = \{v_1, v_2, \ldots, v_{10}\}$ and $E = \{v_1v_2, v_2v_3, v_3v_4, v_4v_5, v_1v_5, v_6v_8, v_8v_{10}, v_{10}v_7, v_7v_9, v_9v_6, v_1v_6, v_2v_7, v_3v_8, v_4v_9, v_5v_{10}\}$. This graph is known as the classic **Petersen** graph, as illustrated in Fig. 2.2a. Figure 2.2b and c provides two different graphical representations of this graph.

A graph G is called a **labeled graph** if each vertex in G is assigned a label; otherwise, it is referred to as a **non-labeled** graph. The graph shown in Fig. 2.2a is a labeled graph, while the graphs in Fig. 2.2b and c are non-labeled graphs.

In the definition of a simple graph $G = (V, E)$, the elements in the edge set E are unique, meaning no two edges can be identical. When this restriction is relaxed, allowing for multiple occurrences of the same edge in E, we introduce the concept of having $i\,(> 1)$ edges connecting a pair of vertices in G. These edges are referred to as **multiple edges** (or **parallel edges**). Graphs that contain multiple edges are classified as **multigraphs**. To further clarify, we can extend the set $V^{(2)}$ into a new set $V_{=}^{(2)}$ by allowing repeated elements. In this context, each element in $V_{=}^{(2)}$ is also an element of $V^{(2)}$ and can appear an arbitrary number of times. Then, the ordered pair $(V, E_{=})$ represents a multigraph and is denoted as $G_{=}$, and the repeated elements in $E_{=}$ are multiple edges, where $E_{=} \subseteq V_{=}^{(2)}$.

Example 2.2 Let $G_{=} = (V, E)$, where $V = \{v_1, v_2, v_3, v_4, v_5\}$ and $E_{=} = \{e_i | i = 1, 2, \ldots, 11\}$ with $e_1 = e_2 = v_1v_5$, $e_3 = e_4 = e_5 = v_1v_2$, $e_6 = v_1v_4$, $e_7 = v_1v_3$,

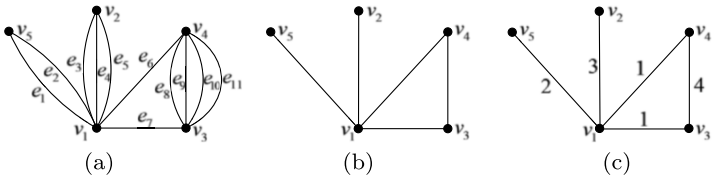
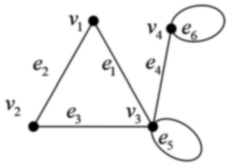


Fig. 2.3 Illustration of multigraphs, basic graphs, and edge-weighted graphs: (**a**) A multigraph $G_=$; (**b**) the basic graph $B(G)$ of $G_=$; (**c**) the edge-weighted graph corresponding to $G_=$

Fig. 2.4 A pseudograph



$e_8 = e_9 = e_{10} = e_{11} = v_3v_4$. Then, $G_=$ is a multigraph; Fig. 2.3a provides a graphical representation of it.

The **basic** graph of a multigraph $G_= = (V, E)$, denoted as $B(G)$, is a simple graph vertex set V, where two vertices are adjacent in $B(G)$ if and only if they are connected by at least one edge in $G_=$. Figure 2.3b provides a graphical representation of the basic graph of the multigraph shown in Fig. 2.3a.

The definitions of simple graphs and multigraphs require that each edge connect two distinct vertices. When this restriction is removed, edges can connect to the same vertex, which we refer to as **loops**. Graphs that contain loops are classified as **pseudographs**. In more detail, we extend the sets $V^{(2)}$ (and $V_=^{(2)}$) to a new set $V_O^{(2)}$, which permits elements from the set $\{\{v_i, v_i\} | i = 1, 2, \ldots, n\}$ to occur an arbitrary number of times, where $V = \{v_1, v_2, \ldots, v_n\}$. Then, the ordered pair (V, E_O) represents a pseudograph, denoted by G_O, where $E_O \subseteq V_O^{(2)}$. The element $\{v_i, v_i\} \in E_O$ for $i \in \{1, 2, \ldots, n\}$ corresponds to a loop.

Example 2.3 Figure 2.4 illustrates a pseudograph $G_O = (V, E_O)$ with $V = \{v_1, v_2, v_3, v_4\}$ and $E_O = \{e_i | i = 1, 2, 3, 4, 5, 6\}$, where $e_5 = v_3v_3$ and $e_6 = v_4v_4$ are loops.

Remark 2.3 Multigraphs must contain multiple edges, and pseudographs must contain loops.

For a multigraph $G_=$, labeling each edge e in its basic graph $B(G)$ with the number of multiple edges corresponding to e in $G_=$ is equivalent to assigning weights to the edges of $B(G)$. For example, in the multigraph illustrated in Fig. 2.3a, the edge-weight function ω, defined based on this method, is $\omega(v_1v_5) = 2$, $\omega(v_1v_2) = 3$, $\omega(v_1v_3) = \omega(v_1v_4) = 1$, and $\omega(v_3v_4) =$; Fig. 2.3c illustrates this weight assignment for edges. More generally, an **edge-weighted graph** G can be represented as a 3-tuple (V, E, ω), where (V, E) constitutes a simple

graph and $\omega : E \rightarrow \mathrm{R}$ is a function that assigns a weight to each edge e_i in the edge set $E = \{e_1, e_2, \ldots, e_m\}$. The collection of weights, represented as $\omega(E) = (\omega(e_1), \omega(e_2), \ldots, \omega(e_m))$, is termed the **edge-weighted vector**. For instance, the edge-weighted vector for the graph shown in Fig. 2.3c is given by $(\omega(v_1v_2), \omega(v_1v_3), \omega(v_1v_4), \omega(v_1v_5), \omega(v_3v_4)) = (3, 1, 1, 2, 4)$.

Edge-weighted graphs have good application backgrounds, such as transportation networks and biological neural networks. In transportation networks, vertices represent cities, edges represent whether there are roads, railways, air routes, or sea routes between two cities, and the weight of the edges represents the distance between the two cities (including road distance, railway distance, flight distance, and sea distance, etc.). In biological neural networks, vertices represent neurons of a biological body (such as a human), edges represent synapses between two neurons, and the edges' weight represents the corresponding synapses' thickness. The human brain has about 10^{12} neurons and $10^{15} \sim 10^{16}$ synapses, but research on the thickness of synapses is relatively less, and in-depth research in this area is crucial for brain science research.

A **vertex-weighted graph** is defined as a 3-tuple (V, E, ω), where (V, E) represents a simple graph and $\omega : V \rightarrow \mathrm{R}$ is a function that assigns a weight to each vertex. The collection of weights, denoted as $\omega(V) = (\omega(v_1), \omega(v_2), \ldots, \omega(v_n))$, is referred to as the **vertex-weighted vector**. In this context, $\omega(v_i)$ (for $i \in \{1, 2, \ldots, n\}$) signifies the **weight** of the vertex v_i, with $V = \{v_1, v_2, \ldots, v_n\}$. For instance, consider the graph shown in Fig. 2.5, which has $V = \{v_1, v_2, \ldots, v_{12}\}$ and the vertex-weighted vector $\omega(V) = (\omega(v_1), \omega(v_2), \ldots, \omega(v_{12})) = (1, 2, 3, 4, 1, 3, 2, 4, 1, 4, 3, 2)$. In this graph, each vertex is assigned a weight of 1, 2, 3, or 4, and it is important to note that the two endpoints of each edge possess different weights. Thus, this type of vertex-weighted assignment can be interpreted as a form of vertex coloring in the graph, where the weight represents the color and the set of colors is {1, 2, 3, 4}. Like edge-weighted graphs, vertex-weighted graphs also have a wide range of applications, including traffic signal light design, scheduling tasks, and route planning problems.

A more universally applicable graph form is a **mixed-weighted graph** (often called a weighted graph). This can be expressed as a 4-tuple $(V, E, \omega_e, \omega_v)$, where (V, E) represents a simple graph, (V, E, ω_e) is an edge-weighted graph, and (V, E, ω_v) is a vertex-weighted graph.

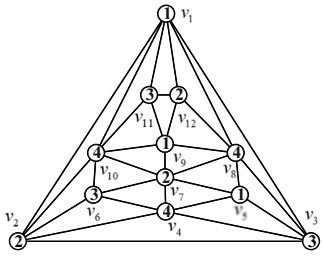


Fig. 2.5 A vertex-weighted graph

In summary, undirected graphs can be categorized into four types, including simple graphs, multigraphs, pseudographs, and weighted graphs. Furthermore, weighted graphs are subdivided into three distinct categories: edge-weighted graphs, vertex-weighted graphs, and mixed-weighted graphs.

Remark 2.4 Unless explicitly stated, all graphs referred to in this book are finite simple undirected graphs.

In the definition of a graph, by changing the unordered pair in $V^{(2)}$ to an ordered pair, one arrives at the concept of a directed graph. Given a non-empty set $V = \{v_1, v_2, \ldots, v_n\}$, we define the set of all 2-element ordered pairs of V as $V^{[2]}$. An ordered pair from v_i to v_j (where $i \neq j$ and $i, j \in \{1, 2, \ldots, n\}$) is represented as (v_i, v_j), which can also be abbreviated as $v_i v_j$. For example, if we have $V = \{v_1, v_2, v_3\}$, the set of all 2-element ordered pairs can be expressed as $V^{[2]} = \{v_1v_2, v_1v_3, v_2v_1, v_2v_3, v_3v_1, v_3v_2\}$. This framework allows us to similarly define a directed graph based on the concept of $V^{[2]}$.

Let V be a non-empty set and $A \subseteq V^{[2]}$. The ordered pair (V, A) is referred to as a **simple directed graph**, denoted as D. In this context, V is called the **vertex set** of D, while A is called the **arc set** of D. The elements within V are termed the **vertices** of D, and the elements in A are called the *arcs* of D. If $a = uv \in A$, then a denotes the arc from u to v, where u serves as the **tail** of the arc and v acts as the **head**. In this context, we can also say that the arc $a = uv$ is **incident** with its tail u and head v; we also say that u is adjacent to v, or that u dominates v.

In a directed graph, a pair of arcs such as uv and vu is known as **symmetric arcs**. Let $D = (V, A)$ denote a directed graph. Its inverse, represented as D', is a directed graph (V', A') where $V' = V$ and $A' = \{uv | uv \in V^{[2]} \text{ and } uv \notin A\}$. The **underlying graph** of the directed graph D, denoted as $U(D)$, is an undirected graph formed by replacing each arc in D with an undirected edge. Figure 2.6 illustrates three representations: (a) the directed graph D, (b) its inverse D', and (c) its underlying graph $U(D)$.

A **completely symmetric directed graph** is a directed graph where each pair of vertices is connected by exactly one pair of symmetric arcs. Figure 2.7 shows the 5-order completely symmetric graph. A **tournament graph** is a directed graph where each pair of vertices is connected by exactly one arc. In other words, a tournament graph is a directed graph whose underlying graph is a complete graph. The tournament graph is known for its good application background. Figure 2.7b shows a 5-order tournament graph.

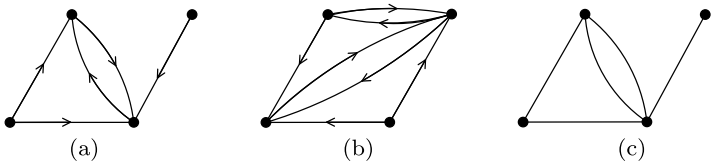


Fig. 2.6 A directed graph, its inverse and underlying graph: (**a**) D; (**b**) D'; (**c**) $U(D)$

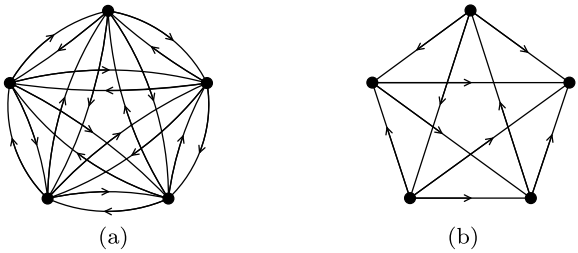


Fig. 2.7 (**a**) The 5-order completely symmetric directed graph and (**b**) the 5-order tournament graph

The directed graphs referenced above are classified as simple directed graphs. Similar to the classification of undirected graphs, directed graphs can be categorized into four types: simple directed graphs, multi-directed graphs, pseudo-directed graphs, and weighted directed graphs. Moreover, weighted directed graphs can be further divided into three categories: arc-weighted directed graphs, vertex-weighted directed graphs, and mixed-weighted graphs.

This book primarily addresses undirected graphs, so this section offers only a brief overview of the key concepts and classifications of directed graphs. For a more thorough exploration of directed graph theory, please refer to reference [2]. Additionally, for in-depth definitions, notation, and foundational theories concerning undirected graphs, please consult references [3, 4].

2.1.2 Degree Sequence of a Graph

Let $G = (V, E)$ be a simple graph. For a vertex $v \in V$, the number of edges incident with v is called the degree of v, denoted as $d_G(v)$, and represented by $d(v)$ when there is no confusion. Obviously,

$$d(v) = |N(v)|. \tag{2.1}$$

The **maximum degree** of the graph G, denoted as $\Delta(G)$, refers to the maximum degree of all vertices in G. The minimum degree of G, denoted as $\delta(G)$, refers to the **minimum degree** of all vertices in G. That is, $\Delta(G) = \max\{d(v)|v \in V\}$ and $\delta(G) = \min\{d(v)|v \in V\}$. If $\Delta(G) = \delta(G) = k$, then G is called a k-**regular** graph. We call a vertex with a degree of zero an **isolated vertex** and a vertex with a degree of 1 a **pendant vertex**.

The **degree sequence** of a graph $G = (V, E)$, denoted as $\pi(G)$, is a sequence that consists of the degrees of all its vertices arranged in a specific order. Generally, this sequence is typically presented in either a monotonically increasing or decreasing format. Let $V = \{v_1, v_2, \ldots, v_n\}$ and let $d_i = d(v_i)$ for $i = 1, 2, \ldots, n$. If the degrees are organized such that $d_1 \geq d_2 \geq \ldots \geq d_n$, then the degree sequence can be expressed as the n-tuple $\pi(G) = (d_1, d_2, \ldots, d_n)$. In certain cases, increasing order is used to represent the degree sequence, which yields

$\pi(G) = (d_n, d_{n-1}, \ldots, d_1)$. For example, the degree sequence of the graph shown in Fig. 2.3b is (1,1,2,2,4) or (4,2,2,1,1).

The following conclusions regarding the degree sequence are evidently valid.

Theorem 2.1 *Let* $G = (V, E)$ *be an* (n, m)*-graph, where with* $V = \{v_1, v_2, \ldots, v_n\}$. *Then,*

(1) $0 \leq d(v_i) \leq n - 1$ *for every* $i \in \{1, 2, \ldots, n\}$;
(2) There exist two distinct $i, j \in \{1, 2, \ldots, n\}$ *such that* $d(v_i) = d(v_j)$;
(3) $\sum_{i=1}^{n} d_i = 2m$.

Proof Note that G is a simple graph. Consequently, each vertex $v \in V$ can be adjacent to at most $n - 1$ other vertices, which substantiates conclusion (1). The proof for conclusion (2) follows from the pigeonhole principle. As for conclusion (3), since each edge contributes two to the total sum of the degrees of all vertices, this conclusion is also valid. □

2.1.3 Graph Operations

Graphs, much like numbers, vectors, and matrices, can be subjected to a range of operations that fulfill distinct functions. As the field of graph theory progresses, the development of additional operations is anticipated. This section aims to introduce some of the fundamental operations that are essential to graph analysis.

2.1.3.1 Subgraphs and Unary Graph Operations

For two graphs G and H, if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$, then H is called a **subgraph** of G; if, in addition, $V(H) \neq V(G)$ or $E(H) \neq E(G)$, then H is called a **proper subgraph** of G. Among the various types of subgraphs, two special and important categories are spanning subgraphs and induced subgraphs, which are defined as follows:

Let H be a subgraph of G. If $V(H) = V(G)$, then H is referred to as a **spanning subgraph** of G. Let $V' \subseteq V(G)$ with $V' \neq \emptyset$. The **subgraph induced by** V', often called a **vertex-induced subgraph**, denoted as $G[V']$, consists of the vertex set V' and the edge set $\{uv \mid u, v \in V' \text{ and } uv \in E(G)\}$. Similarly, we can define an edge-induced subgraph. Let $E' \subseteq E(G)$ represent a non-empty subset of the edges of a graph G. The **subgraph induced by** E', or simply an **edge-induced subgraph**, denoted as $G[E']$, is formed with the vertex set $\{v \mid v \text{ is an endpoint of an edge in } E'\}$ and the edge set E'.

Next, we introduce two basic operations associated with a graph: the operations of vertex deletion and edge deletion.

Vertex Deletion Let $G = (V, E)$ be a graph, and let $V' \subseteq V$ be a subset of its vertices. **Deleting V' from G** involves removing the vertices in V' along with all edges that are incident to these vertices. The resulting graph is denoted as $G - V'$ and is called the **vertex-deleted subgraph** of G. In fact, $G - V'$ is the subgraph induced by the remaining vertices $V - V'$; thus, we have $G - V' = G[V - V']$. When V' consists of a single vertex, say $\{v\}$, we simply denote $G - \{v\}$ by $G - v$.

Edge Deletion Let $E' \subseteq E$ be a subset of edges from a graph $G = (V, E)$. **Deleting E' from G** entails removing all edges included in E', resulting in a subgraph known as the **edge-deleted subgraph** of G, denoted by $G - E'$. When E' consists of a single edge, denoted as e, we simply denote $G - \{e\}$ by $G - e$.

The operations of vertex deletion and edge deletion are categorized as **unary graph operations** within the field of graph theory, as they pertain to actions taken on a single graph. Figure 2.8 provides an illustration of the concepts of spanning and induced subgraphs, as well as these two types of unary operations.

Based on the concept of subgraphs, we present the definitions of connected graphs and trees. Let $G = (V, E)$ be a graph. If, for every pair of vertices $u, v \in V$, there exists a path that connects u to v within G, then G is defined as a **connected graph**. Consider a subgraph H of G. If H is connected and is not a proper subgraph of any connected subgraph of G, then H is termed a **connected component** of G. Furthermore, if G does not contain any subgraphs that form a cycle, it is classified as **acyclic**. An acyclic graph is also called a **forest**, while a connected acyclic graph is known as a **tree**.

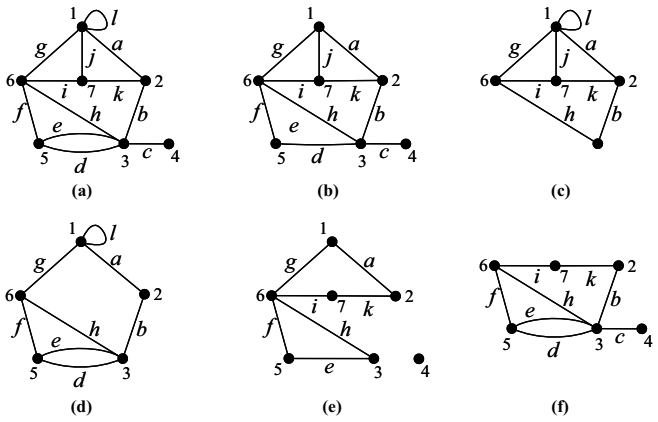


Fig. 2.8 Examples of various subgraphs: (**a**) A graph G; (**b**) a spanning graph of G; (**c**) $G - \{4, 5\}$; (**d**) $G[\{1, 2, 3, 5, 6\}]$; (**e**) $G - \{b, c, d, j, \ell\}$; (**f**) $G[\{b, c, d, e, f, h, i, k\}]$

2.1.3.2 Binary Graph Operations

This section presents an overview of the five primary operations utilized between two graphs, referred to as **binary graph operations**. These operations include union, intersection, difference, symmetric difference, and join. For the purposes of this discussion, we denote the two graphs as $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$.

Union The **union** of two graphs G_1 and G_2, denoted as $G_1 \cup G_2$, is defined as a graph consisting of the vertex set $V_1 \cup V_2$ and the edge set $E_1 \cup E_2$. When G_1 and G_2 have no edges in common, the union $G_1 \cup G_2$ is termed the **direct sum** of G_1 and G_2. Thus, when the book refers to the direct sum operation, it clarifies that the graphs involved do not share any edges.

Intersection The **intersection** of two graphs G_1 and G_2, denoted as $G_1 \cap G_2$, is defined as a graph consisting of the vertex set $V_1 \cap V_2$ and the edge set $E_1 \cap E_2$.

Difference The **difference** between the graphs G_1 and G_2, represented as $G_1 - G_2$, defines the subgraph of G_1 obtained by removing the edges present in G_2 from G_1. Specifically, this can be expressed as $G_1 - G_2 = G_1 - (E_1 \cap E_2)$.

Symmetric Difference The **symmetric difference** of two graphs G_1 and G_2, denoted as G_1
$textcircled{+}G_2$, is the difference between $G_1 \cup G_2$ and $G_1 \cap G_2$, i.e., G_1
$textcircled{+}G_2 = (G_1 \cup G_2) - (G_1 \cap G_2) = (G_1 - G_2) \cup (G_2 - G_1)$.

Figure 2.9 illustrates the definitions of the operations described above.

Join The **join** of two vertex-disjoint graphs G and H, denoted as $G + H$, is the graph with the vertex set $V_1 \cup V_2$ and the edge set $E_1 \cup E_2 \cup \{uv \mid u \in V_1 \text{ and } v \in V_2\}$.

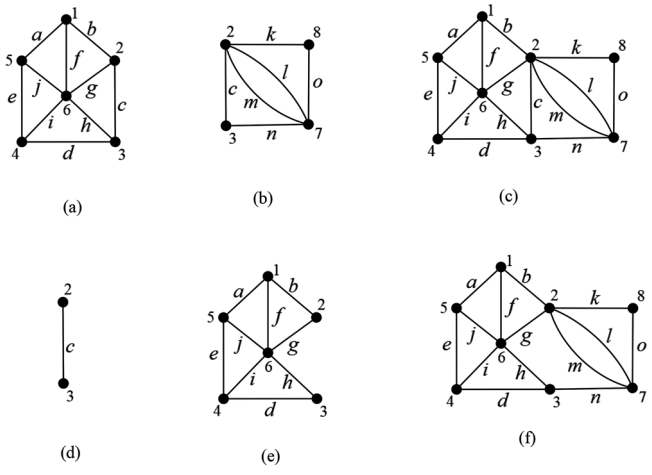


Fig. 2.9 Illustrations for the operations of union, intersection, difference, symmetric difference, and join. (**a**) a graph G_1; (**b**) a graph G_2; (**c**) $G_1 \cup G_2$; (**d**) $G_1 \cap G_2$; (**e**) $G_1 - G_2$; (**f**) $G_1 \oplus G_2$

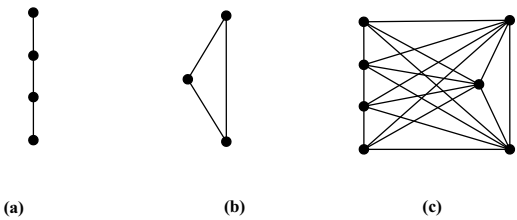


Fig. 2.10 Illustrations for the operation of join: (**a**) P_4; (**b**) C_3; (**c**) $P_4 + C_3$

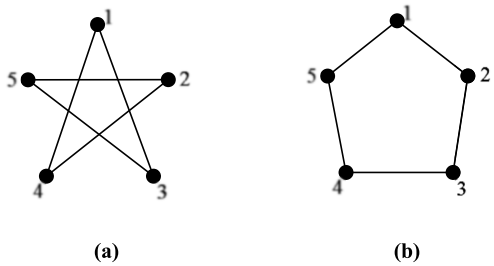


Fig. 2.11 Illustration for the complement operation: (**a**) A graph G; (**b**) $\overline{G}$

In this definition, the edges uv connect every vertex in G to every vertex in H. Figure 2.10 illustrates this idea by showing the join of the 3-length path P_4 and the 3-cycle C_3.

2.1.3.3 Other Unary Graph Operations

In addition to the vertex and edge deletion operations, many other unary graph operations exist. This section introduces two additional operations commonly applied to a single graph: the complement operation and the contraction operation.

Complement Let $G = (V, E)$ be a simple graph. The **complement** of G, denoted as $\overline{G}$, is defined as the graph that has the vertex set V and the edge set $\{uv \mid u, v \in V \text{ and } uv \notin E\}$. Figure 2.11 illustrates a graph and its complement. It can be readily demonstrated that a simple graph G of order n and its complement $\overline{G}$ together form a complete graph of order n. As a result, we obtain the following relationship:

$$|E(G)| + |E(\overline{G})| = \frac{n(n-1)}{2}.$$

Remark 2.5 A graph is complete if and only if its complement is a null graph.

The relationship between a graph and its complement has significant network and system analysis implications. Typically, as a network's structure's complexity grows, its complement structure becomes simpler. By investigating the complement network, we can uncover valuable insights into the characteristics of the original network.

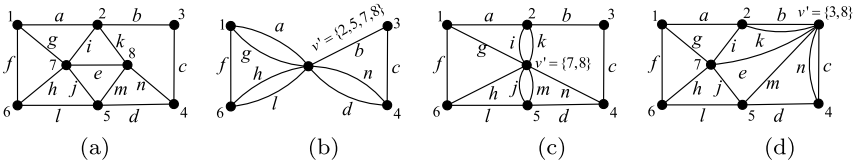


Fig. 2.12 Illustration for the contraction operation: (**a**) A graph G; (**b**) $G \circ \{2, 5, 7, 8\}$; (**c**) $G \circ 78$; (**d**) $G \circ \{3, 8\}$

Contractions Given a simple graph $G = (V, E)$ and a subset of its vertices $V' \subseteq V$. The process of **contracting** V' involves first removing all edges that connect both endpoints in V', and then replacing V' with a single new vertex that is incident with all edges that were originally incident with any vertex in V'. This operation is referred to as the **vertex-contracted operation** concerning V', and the resulting graph is known as the **contraction graph** of G with respect to V', denoted as $G \circ V'$. Furthermore, when $V' = \{u, v\}$ and $uv \in E$, the contraction graph $G \circ V'$ specifically represents the graph obtained by contracting the edge uv in G, denoted as $G \circ uv$. This procedure is also referred to as the **edge-contracted operation**. Fig. 2.12 depicts a graph along with the contraction graphs with respect to the vertex subset $\{2, 5, 7, 8\}$, the edge 78, and the vertex subset $\{3, 8\}$.

2.1.4 Graph Isomorphism

A graph is defined as a structure that illustrates the adjacency between pairs of vertices within a vertex set. The arrangement of the vertices in the graph is irrelevant, and the edges connecting two vertices are not influenced by factors such as length, curvature, or shape. Consequently, some graphs that may initially appear distinct can actually be identical. This relationship between seemingly different graphs is referred to as isomorphism, which is defined as follows:

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ represent two simple graphs. The graphs G_1 and G_2 are considered **isomorphic**, denoted as $G_1 \cong G_2$, if there exists a bijection σ that maps V_1 to V_2 such that for any $u, v \in V_1$, u and v are adjacent in G_1 if and only if $\sigma(u)$ and $\sigma(v)$ are adjacent in G_2. This bijection σ is referred to as an **isomorphic mapping** between G_1 and G_2. The graphs G and G' depicted in Fig. 2.13 are isomorphic to one another, with the isomorphic mapping σ defined as follows: $\sigma(v_i) = i$ for $i = 1, 2, 3, 4, 5, 6$, where $V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ and $V(G') = \{1, 2, 3, 4, 5, 6\}$.

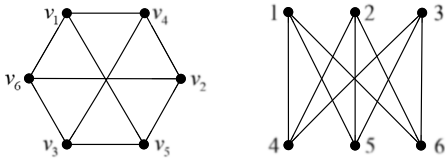


Fig. 2.13 Two classic isomorphic graphs

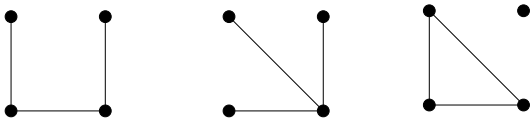


Fig. 2.14 Three non-isomorphic (4,3)-graphs

2.1.4.1 Isomorphism Testing Algorithms

The task of determining whether two graphs are isomorphic is referred to as the **graph isomorphism testing problem**. The classification of this problem as a **P** problem or an **NP**-complete problem remains an open question. Below, several intuitive methods for addressing the graph isomorphism problem will be discussed.

Theorem 2.2 (Order and Size-Based Method) *If two graphs are isomorphic, they have the same order and size; however, the reverse is not necessarily true.* □

This method (Theorem 2.2) only allows for the determination of non-isomorphism between two graphs under specific conditions: if the number of vertices in the two graphs is not equal, they must be non-isomorphic. Similarly, if the number of vertices is the same but the number of edges differs, the graphs are also non-isomorphic. However, this approach is somewhat simplistic, as numerous graphs can share the same number of vertices and edges while still being non-isomorphic. For example, there are three non-isomorphic (4,3)-graphs, as depicted in Fig. 2.14. Next, we present the method of comparing degree sequences.

Theorem 2.3 (Degree Sequence-Based Method) *Isomorphic graphs have identical degree sequences; however, the reverse is not necessarily true.* □

This method (Theorem 2.3) can assess whether two graphs are non-isomorphic by examining their degree sequences. If the degree sequences differ, the graphs are confirmed to be non-isomorphic. However, this approach has limitations, as several non-isomorphic graphs can possess the same degree sequence. For instance, there are two graphs with six vertices that share the degree sequence (2, 2, 2, 2, 2, 2). One graph is the 6-cycle C_6, while the other consists of two disjoint triangles.

Theorem 2.4 (Complement Graph-Based Method) *Two graphs are isomorphic if and only if their complements are isomorphic.* □

Generally, determining whether two graphs are isomorphic is typically a challenging problem, and various algorithms have been created to tackle it. Currently, the most effective algorithm for graph isomorphism is the quasi-polynomial time

algorithm introduced by Babai [5] in 2016. For a more in-depth exploration of graph isomorphism algorithms, please consult the review referenced in [6].

2.1.4.2 Applications of Graph Isomorphism

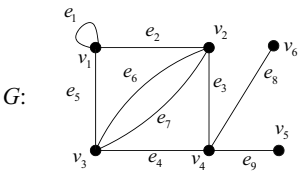
The graph isomorphism problem is highly relevant in several practical applications. Assessing whether two systems are isomorphic is crucial for effective system modeling in system engineering. Moreover, it is important to eliminate numerous isomorphic graphs when generating graphs using computer algorithms. For instance, isomorphic algorithms have been used to identify subgraphs in the study of Ramsey number theory [7, 8].

2.1.5 Matrices of Graphs

An invariant of a graph is a numerical value or vector that remains unchanged for any isomorphic graph. For instance, the number of vertices in a graph is an invariant. Likewise, the number of edges is also an invariant. Additionally, the chromatic number and the independence number are also invariants of a graph. The degree sequence is a vector-type invariant. In general, it is difficult for graph invariants to accurately characterize a graph's structure and features.

The matrices associated with graphs play a crucial role in accurately representing their structure and characteristics, forming the foundation for computer-based information processing of graphs. This section offers an overview of two primary matrices in graph theory, incidence and adjacency matrices, and another significant matrix in algebraic graph theory—the Laplacian matrix.

Let $G = (V, E)$ be an (n, m)-graph, where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$. The **incidence matrix** of G, denoted as $M(G) = (m_{ij})_{n\times m}$, is an $n \times m$ matrix. Each entry m_{ij} (with $i = 1, 2, \ldots, n$ and $j = 1, 2, \ldots, m$) indicates the number of times vertex v_i is incident to edge e_j, taking on values of 0, 1, or 2. Figure 2.15 visually represents a graph and its corresponding incidence matrix.



$$M(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccccccc} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 & e_8 & e_9 \end{array} \\ \begin{bmatrix} 2 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Fig. 2.15 A graph G and its incidence matrix $M(G)$

The incidence matrix is a matrix that represents the structure of a graph by depicting the relationships between its vertices and edges. When a graph has a significant number of edges, the storage space required for computations using the incidence matrix can be quite substantial. As a result, the adjacency matrix is often used as an effective alternative. The adjacency matrix illustrates the adjacency relationships among the graph's vertices.

Let $G = (V, E)$ be an n-order graph, where $V = \{v_1, v_2, \ldots, v_n\}$. The **adjacency matrix** of G, denoted by $A(G) = (a_{ij})_{n\times n}$, is an $n \times n$ matrix. Each entry a_{ij} (with $i, j = 1, 2, \ldots, n$) indicates the number of edges connecting v_i and v_j. For example, the adjacency matrix corresponding to the graph shown in Fig. 2.15 is as follows:

$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 2 & 1 & 0 & 0 \\ 1 & 2 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

The adjacency matrix $A(G)$ of a graph G has the following properties:

(1) The matrix $A(G)$ is symmetric, and the diagonal elements of $A(G)$ are zero if and only if the graph G is acyclic.
(2) The matrix $A(G)$ is a symmetric 0-1 matrix with diagonal elements equal to zero if and only if the graph G is a simple graph.

Remark 2.6 Simple graphs have a one-to-one correspondence with 0-1 symmetric matrices in which all diagonal elements are equal to 0. This relationship arises because, for any 0-1 symmetric matrix A with zero diagonal elements, one can uniquely construct a simple graph G such that $A(G) = A$. Conversely, the conclusion is supported by the property (2) of the adjacency matrix.

Remark 2.6 illustrates the applicability of matrices in the analysis of graph properties. Below, two fundamental theorems (Theorems 2.5 and 2.6) are presented, and their respective proofs can be found in reference [3].

Theorem 2.5 *Let G be a simple graph with order p and $A(G) = (a_{ij})_{p\times p}$ is the adjacency matrix of G. Then,*

(1) In $(A(G))^\ell$, the ℓ-th power of the adjacency matrix $A(G)$, the entry $a_{ij}^{(\ell)}$ corresponding to the i-th row and j-th column indicates the number of walks[2] of length ℓ from vertex v_i to vertex v_j in the graph G, where $i, j \in \{1, 2, \ldots, p\}$.

[2] A walk refers to a sequence of vertex-edge interactions: $v_0e_1v_1e_2\ldots e_kv_k$, where k is called the length of the walk, and $e_i = v_{i-1}v_i$, $i = 1, 2, \ldots, k$.

(2) For $i \in \{1, 2, \ldots, p\}$, *we have* $a_{ii}^{(2)} = \sum_{j=1}^{p} a_{ij}a_{ji} = d(v_i)$.

(3) For $i \in \{1, 2, \ldots, p\}$, *the entry* $a_{ii}^{(3)}$ *corresponds to twice the number of triangles in G that include vertex* v_i. □

The following theorem characterizes the relationship between the adjacency matrix and the incidence matrix in regular graphs.

Theorem 2.6 *Let* $M(G)$ *and* $A(G)$ *be the incidence matrix and adjacency matrix of a k-regular graph G of order n, respectively. Then,*

$$M(G)M^T(G) = A(G) + kI_n,$$

where $M^T(G)$ *is the transpose of* $M(G)$ *and* I_n *is a n-order identity matrix.* □

Let $G = (V, E)$ be a graph of order n, where $V = \{v_1, v_2, \ldots, v_n\}$. The $n \times n$ matrix $L(G) = D(G) - A(G)$ is referred to as the Laplacian matrix of G. In this context, $A(G)$ denotes the adjacency matrix of the graph, while $D(G)$ is defined as follows:

$$D(G) = \begin{bmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{bmatrix}$$

In this matrix, the diagonal elements d_i (for $i = 1, 2, \ldots, n$) represent the degree of the vertex v_i in the graph, while all off-diagonal elements are zeros.

For a graph $G = (V, E)$, the Laplacian matrix $L(G)$ possesses the following properties:

(1) $L(G)$ is a symmetric and semi-positive definite matrix.
(2) The rank of $L(G)$ is $n - k$, where k represents the number of connected components in the graph G.
(3) For any vector $X = (x_1, x_2, \ldots, x_n)^T$, the following holds:

$$X^T L(G)X = \sum_{v_i v_j \in E} (x_i - x_j)^2.$$

(4) The sum of each row and each column in $L(G)$ is zero.
(5) The algebraic cofactors corresponding to any element in $L(G)$ are identical.

2.1.6 Graph Coloring

Graph coloring is a significant field within graph theory that has a variety of applications, such as task scheduling, process optimization, protein structure prediction, and code-breaking. One of the main goals of this book is to develop a DNA computing approach to address the graph coloring problem, along with a specialized DNA computer model. This section presents a comprehensive introduction to the graph coloring problem.

2.1.6.1 Definition and Classification

A k-**vertex-coloring** of a graph $G = (V, E)$, commonly referred to as k-**coloring**, is a mapping f from the vertex set V to the color set $C(k) = \{1, 2, \ldots, k\}$ such that for any edge $xy \in E$, the colors assigned to the vertices at either endpoint of the edge must differ, meaning $f(x) \neq f(y)$. A graph G is called k-**colorable** if G admits a k-vertex-coloring. The **chromatic number** of a graph G, denoted as $\chi(G)$, represents the smallest positive integer k for which G is k-colorable. If $\chi(G) = k$, then G is called a k-chromatic graph.

Each k-coloring f of a graph $G = (V, E)$ uniquely corresponds to a k-**color class partition** $(V_1, V_2, \ldots, V_k)$ that satisfies the following conditions: $V = V_1 \cup V_2 \cup \ldots \cup V_k$ and $V_i \neq \emptyset$ for each i, where $i = 1, 2, \ldots, k$. In this context, V_i represents the subset of vertices assigned color i, known as a **color class**. If the colors of the vertices in V_i and V_j are swapped while the colors of all other vertices remain unchanged, the resulting coloring is deemed **equivalent** to the original. As a result, each equivalence class of a k-coloring contains $k!$ distinct colorings. The collection of all k-colorings, obtained by selecting one k-coloring from each equivalence class, is denoted as $C_k^0(G)$. Let $f \in C_k^0(G)$. We denote the subgraph induced by the vertices colored with colors i or j under f as G_{ij}^f. This is referred to as a **bicolor-induced subgraph**, where $i, j \in C(k)$ and $i \neq j$. When the context permits, we may simply represent G_{ij}^f as G_{ij}. The connected components within G_{ij} are referred to as ij-**components** or **bicolor-components**.

If $C_k^0(G)$ contains only one k-coloring, then G is called a uniquely k-colorable graph. The graph shown in Fig. 2.16 is an example of uniquely 3-colorable graphs.

A k-**edge-coloring** of a graph $G = (V, E)$, commonly referred to as **edge coloring**, involves assigning colors from the color set $C(k) = \{1, 2, \ldots, k\}$ to the

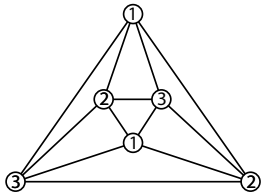


Fig. 2.16 A uniquely 3-colorable graph of order six

edges in E such that any two adjacent edges e and e' receive different colors, i.e., $f(e) \neq f(e')$. If a graph G has a k-edge-coloring, it is termed k-**edge-colorable**. The **edge chromatic number** of the graph G, denoted as $\chi'(G)$, represents the smallest positive integer k for which G is k-edge-colorable. When $\chi'(G) = k$, the graph G is called a k-edge-chromatic graph.

In a manner analogous to vertex-coloring, each k-edge-coloring of the graph G corresponds to a partition $\{E_1, E_2, \ldots, E_k\}$ of its edge set E. Each E_i (for $i = 1, 2, \ldots, k$) is referred to as an **edge-color class**, representing the collection of all edges assigned the color i. In other words,

$$E(G) = \bigcup_{i=1}^{k} E_i,\ E_i \neq \emptyset,\ E_i \cap E_j = \emptyset,\ i \neq j, i, j = 1, 2, \ldots, k,$$

A k-**total-coloring** of a graph $G = (V, E)$, commonly referred to as **total coloring**, is a mapping from the set $V \cup E$ to the color set $C(k) = \{1, 2, \ldots, k\}$ such that any pair of incident or adjacent elements x and y within $V \cup E$ receive distinct colors, i.e., $f(x) \neq f(y)$. When a k-total-coloring exists for G, the graph G is called a k-**total-colorable** graph. The **total chromatic number** of graph G, denoted as $\chi^T(G)$, indicates the smallest positive integer k that allows G to be k-total-colorable.

2.1.6.2 Chromatic Number of a Graph

In the previous section, we examined various types of graph colorings along with their associated chromatic numbers. A pertinent question arises: how can we ascertain the chromatic number of a graph? Furthermore, is there a connection between the chromatic number and other graph invariants?

Theorem 2.7 ([9]) *The chromatic number of any graph G satisfies the inequality $\chi(G) \leq \Delta(G)+1$, where $\Delta(G)$ represents the maximum degree of the graph G.* □

Theorem 2.7 establishes an upper bound for the chromatic number of a graph. In addition, for a graph G that is either a complete graph or a cycle of odd length, it is true that $\chi(G) = \Delta(G) + 1$. For graphs of different types, this conclusion can be further enhanced.

Theorem 2.8 (Brooks' Theorem [9]) *If the graph G is neither an odd cycle nor a complete graph, then $\chi(G) \leq \Delta(G)$.* □

Note that a k-vertex-coloring of a graph $G = (V, E)$ $(E \neq \emptyset)$ corresponds to a partition of the vertex set V into k color classes. Therefore, when $k = 2$, the vertex set V of the nonempty graph G can be divided into two independent sets, V_1 and V_2. This implies that any edge in E connecting a vertex in V_1 must connect to a vertex in V_2 and vice versa. Such a graph G is known as a **bipartite graph**.

Theorem 2.9 *The chromatic number of a graph G is equal to two if and only if G is a nonempty bipartite graph.* □

Theorem 2.10 (Vizing's Theorem [10, 11]) *For any simple graph G, it holds that* $\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$. □

Vizing's theorem provides a notable conclusion, as there exist graphs for which $\chi'(G) = \Delta(G)$, as well as graphs for which $\chi'(G) = \Delta(G) + 1$. Despite this, the question of what type of graph G meets the condition $\chi'(G) = \Delta(G)$ remains an open problem.

With regard to the total chromatic number, Behzad [12] and Vizing [13] independently introduced the renowned Total Coloring Conjecture, which has yet to be resolved to this day.

Conjecture 2.1 (Total Coloring Conjecture [12, 13]) For any simple graph G, the following inequality holds: $\Delta(G) \leq \chi^T(G) \leq \Delta(G) + 1$.

2.1.6.3 Vertex Coloring Algorithms

Vertex-coloring algorithms are employed to determine the chromatic number of a graph or to create a k-coloring for a k-chromatic graph. Both of these tasks are classified as **NP**-complete problems. This section highlights a selection of established algorithms for vertex coloring.

(1) Vertex Contraction and Edge Addition Method [14]
Let $G = (V, E)$ be a graph, and let $u, v \in V$ such that $uv \notin E$. We denote the graph formed by adding the edge uv to G as $G + uv$. Furthermore, $G \circ \{u, v\}$ represents the graph obtained by contracting the vertex subset $\{u, v\}$ in G.

Theorem 2.11 *Let u, v be two nonadjacent vertices of a graph G. Then* $\chi(G) = \min\{\chi(G + uv), \chi(G \circ \{u, v\})\}$. □

According to Theorem 2.11, the vertex contraction and edge addition method serves as an algorithm for vertex coloring.

Consider a graph $G = (V, E)$ with two nonadjacent vertices, u and v. The operation of contracting the pair {u, v} in G is termed **vertex contraction**, while adding an edge between u and v is termed **edge addition**. By repeatedly applying vertex contraction and edge addition, the graph can be transformed into some complete graphs. The chromatic number of the graph G corresponds to the chromatic number of the smallest complete graph. Figure 2.17 illustrates this process for calculating the chromatic number of the 5-cycle C_5.

(2) Independent Set Method [15]
Let S be an independent set of a graph $G = (V, E)$. If each vertex in $V - S$ is adjacent to at least one vertex in S, then S is referred to as a **maximal independent set** of G. In addition, if S is an independent set of G and no other independent set S' exists such that $|S'| > |S|$, then S is known as the **maximum independent set**

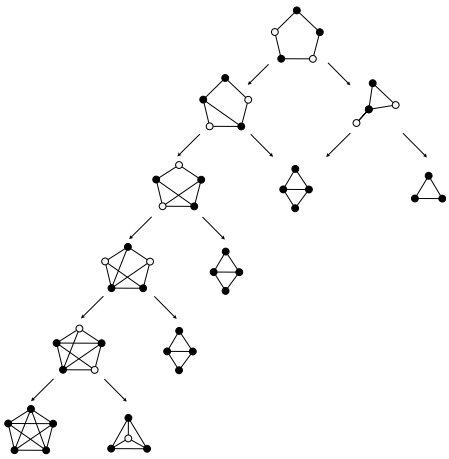


Fig. 2.17 An illustration for determining the chromatic number of the 5-cycle C_5 by vertex contraction and edge addition method

of G. A **cover** of the graph G is defined as a subset $K \subseteq V$ such that every edge in G has at least one endpoint in K. A **minimal cover** of G is a vertex subset K that satisfies the condition: for each vertex v in V, either $v \in K$ or $N(v) \subseteq K$, where G contains no isolated vertex.

Since the vertices in an independent set can all be assigned the same color, the vertex set of a graph can be partitioned into subsets, with each subset forming an independent set. This partitioning is known as an **independent partition**, and the total number of independent sets created is called the **independent partition number**. Thus, the minimum independent partition number of the graph's vertex set corresponds to the graph's chromatic number. Consequently, the chromatic number can be determined by examining all possible independent partitions. Furthermore, since each independent set is a subset of some maximal independent set, the following recursive formula holds:

$$\chi(G) = 1 + \min\{\chi(G - I) \mid I \text{ is a maximal independent set of } G\}.$$

Consequently, to determine the chromatic number of a graph, we can recursively enumerate all maximal independent sets. Importantly, the complement of each maximal independent set corresponds to a minimal cover. We can identify maximal independent sets by determining minimal covers.

Let v be a vertex of a graph G. In the process of finding a minimal cover, there are two choices: you can either select vertex v or choose all the vertices in $N(v)$. This process can be translated into an algebraic expression. In this notation, "selecting vertex v" is denoted simply as v. The symbol "+" indicates "or," while "$\times$" signifies "and." For convenience, we will use uv to represent $u \times v$. Considering the graph shown in Fig. 2.18, we apply this method to calculate its chromatic number.

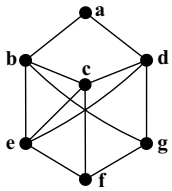


Fig. 2.18 A graph with order seven

According to the established conventions, the corresponding algebraic expression for the minimal cover of this graph is:

$$(a + bd)(b + aceg)(c + bdef)(d + aceg)(e + bcdf)(f + ceg)(g + bdf)$$
$$= aceg + bcdeg + bdef + bcdf.$$

It is crucial to recognize that, since we are focused solely on the minimal cover, the outcome of the previously mentioned formula has eliminated non-minimal terms, i.e., those that are proper subsets of other terms. For instance, the set $\{a, c, e, g\}$ is a proper subset of $\{a, b, c, d, e, f, g\}$, which is why the term $abcdefg$ has been excluded. Consequently, we can infer that $\{a, c, e, g\}$, $\{b, c, d, e, g\}$, $\{b, d, e, f\}$, and $\{b, c, d, f\}$ represent all the minimal covers of Fig. 2.18, while their complements, $\{b, d, f\}$, $\{a, f\}$, $\{a, c, g\}$, and $\{a, e, g\}$, correspond to all the maximal independent sets. We remove these four maximal independent sets from the graph by applying the previously stated chromatic number recursion formula. We will continue identifying and eliminating maximal independent sets from the current graph until all vertices have been removed. Through this method, we determine that the chromatic number of the graph is 3, and the corresponding 3-coloring can be expressed as follows: $\{b, d, f\}$, $\{a, e, g\}$, $\{c\}$.

(3) Powell Method [16]
The specific steps of the Powell method for graph coloring are outlined below:

Step 1. Arrange the vertices of the graph in descending order based on their degree. (For vertices that have the same degree, the order can be determined arbitrarily.)
Step 2. Begin by coloring the first vertex with the initial color (let's designate it as Color 1). Then, proceed to color the first uncolored vertex that is not adjacent to any of the previously colored vertices with Color 1. Continue this process until no more vertices meet these criteria.
Step 3. Repeat Step 2 for the remaining uncolored vertices, applying the second color (designated as Color 2), then the third color (Color 3), and so on, until all vertices have been colored.

Using Fig. 2.18 as an illustration, we begin by arranging the vertices in descending order based on their degree: c, e, b, d, f, g, a. We start by assigning Color 1

to vertex c. Next, we color vertices g and a, which are not adjacent to c, with the same color, Color 1. Following this, we color vertex e with Color 2. We examine its adjacent vertices: b, d, and f. Since g and a have already been assigned Color 1, only vertex e requires Color 2. We then move on to vertex b and color it with Color 3. Upon checking its adjacent vertices, we find that both vertex d and vertex f should also be assigned Color 3. At this stage, all vertices have been colored, and the chromatic number of the graph is three.

In addition to the above methods, there are also many other well-established algorithms for vertex coloring in graphs, including sequential coloring, the enhanced SEQ algorithm, and the DSATUR algorithm, among others. For those interested in exploring vertex coloring algorithms based on bionic and biological computing, we recommend consulting references [17, 18].

2.1.6.4 Application of Graph Coloring

The graph coloring problem is not only of considerable theoretical interest but also has numerous practical applications. This section highlights a few of these applications.

(1) Traffic Signal Light Problem
At intersections, traffic signal lights are essential for managing vehicle flow. When cars from two different lanes attempt to enter the intersection at the same time, the potential for collisions arises; therefore, vehicles from these lanes cannot safely proceed simultaneously. It is crucial to determine the minimum number of signal phases required for the traffic light to ensure safety.

This problem can be addressed through vertex coloring of a graph. We begin by constructing a graph in which the vertex set represents all lanes at the intersection and two vertices are considered adjacent if the vehicles in the corresponding lanes cannot enter the intersection concurrently. Thus, the task of determining the minimum number of signal phases required is equivalent to finding the number of colors necessary for the graph.

(2) Storage Problem
A chemical product company is faced with the challenge of storing various chemical products, some of which cannot come into contact with one another due to safety risks or potential deterioration. As a result, the company must organize its warehouse in such a way that incompatible products are stored in separate areas. The key question is: what is the minimum number of sub-warehouses needed to achieve this?

This scenario can be reinterpreted as a vertex coloring problem. We can construct a graph where the vertices represent all types of chemical products to be stored. Two vertices are adjacent if the corresponding products cannot touch each other. Thus, the minimum number of sub-warehouses required corresponds to the chromatic number of the graph.

(3) Class Scheduling Problem
When scheduling classes for a group of students in a school, the initial step is to clarify the class hours for each teacher assigned to each class. The objective is to develop a timetable that minimizes the total number of class hours needed. This class scheduling problem can be formulated as a graph edge coloring problem. We construct a bipartite graph G with a vertex set $V = \{t_1, t_2, \ldots, t_m, c_1, c_2, \ldots, c_n\}$, where the set of teachers is $\{t_1, t_2, \ldots, t_m\}$ and the set of classes is $\{c_1, c_2, \ldots, c_n\}$. If teacher t_i is responsible for teaching p_{ij} lessons for class c_j, then there are p_{ij} edges linking the vertices t_i and c_j. Therefore, finding a solution that minimizes the number of class hours is equivalent to determining the edge chromatic number of the graph.

In practical applications, the class scheduling problem is often complicated by various constraints, such as a limited number of available classrooms. For further exploration of related research on this topic, please refer to references [19, 20].

2.2 Turing Machine

The Turing machine is a mathematical model of an electronic computer. It is the "soul" of an electronic computer. The Turing machine has made an indelible contribution to the development of human social civilization. Since the **NP**-complete problem is defined under the Turing machine, in order to conduct in-depth research on the **NP**-complete problem, this section first introduces the origin, definition, basic theory of the Turing machine, and the complexity of solving problems under the Turing machine.

2.2.1 The Founder of the Turing Machine: Turing

The person who proposed the Turing machine is Alan Turing; this section will first provide a brief introduction to this great genius scientist. Turing was born in London, England, in 1912, began majoring in mathematics at the University of Cambridge in 1931, and graduated with a bachelor's degree in 1934 with excellent grades. In 1935, he continued to pursue a master's degree at the University of Cambridge and, during this period, proposed the concept of the Turing machine. In 1936, Turing went to Princeton University in the United States for further study under the guidance of Alonzo Church and obtained a doctorate in mathematics in 1938. During Turing's doctoral studies, John von Neumann was already a professor of quantum mechanics at Princeton University. He was very interested in the Turing machine proposed by Turing and invited him to be his assistant, but Turing politely declined and returned to work in the UK after graduation. Based on the Turing machine and basic electronic components, von Neumann established the architecture of today's electronic computers. Turing's impact on the development of theoretical computer

science is enormous, and he is considered the father of theoretical computer science. Hence, the highest award in the field of computer science is named the **Turing Award**. The Turing machine was invented by Turing, and the origin of the Turing machine is introduced below.

In his 1936 paper "On Computable Numbers, with an Application to the Entscheidungsproblem" [21], Turing proposed a concept of computability similar to that of Alonzo Church and proved that Hilbert's Entscheidungsproblem is unsolvable. In the process of proving, Turing defined several types of computing machines, one of which is the predecessor of the Turing machine. This paper is considered one of the most influential mathematical papers in history.

In fact, the study of "computability" can be traced back to 1904, when David Hilbert proposed studying mathematical proofs themselves as mathematical objects. In 1922, he proposed his proof theory research plan at a conference in Hamburg, Germany, known as the Hilbert Program. According to this plan, specific mathematical theories and the logic used are axiomatized at the same time to form a formal system. The Hilbert Program presented an exciting endeavor, with the goal of "once and for all eliminating any doubts about the reliability of the mathematical foundation". In addition to Hilbert, this endeavor attracted a group of young mathematicians, such as Wilhelm Ackermann and John von Neumann. In 1931, Kurt Gödel proved that any formal system that includes first-order predicate logic and elementary number theory is incomplete, thereby negating the Hilbert Program. From the spring of 1935 to the spring of 1936, Turing and Alonzo Church began to study the decidability of problems from Gödel's incompleteness theorem. On May 28, 1936, Turing published "On Computable Numbers, with an Application to the Entscheidungsproblem" in two parts in the Journal of the London Mathematical Society, in which he restated Gödel's achievements in 1931 and proposed the concepts of deterministic Turing machines, non-deterministic Turing machines, and universal Turing machines. The Turing machine later became the foundation of computability theory and computational complexity theory.

In 1938, in his doctoral thesis "Systems of Logic Based on Ordinals" [22], Turing introduced the concepts of ordinal logic and relative computation, expanding the research field of mathematical logic. In addition to pure mathematical work, Turing also studied cryptography and built an electromechanical binary multiplier. During World War II, Turing worked at the Government Code and Cypher School at Bletchley Park, the UK's code-breaking center, which produced extremely important intelligence. Turing designed technology to accelerate the decryption of German codes, built an electromechanical device that could decrypt messages encrypted by the Enigma machine, and played a key role in decrypting intercepted information, enabling the Allies to defeat the Axis in several key battles, including the Battle of the Atlantic. After the war, Turing worked at the National Physical Laboratory in the UK, where he designed the Automatic Computing Engine, one of the earliest stored-program computers. In 1948, Turing joined Max Newman's computer laboratory at the University of Manchester, helped develop the Manchester computer, and became interested in mathematical biology. Subsequently, Turing's research involved biological computation and pattern formation, which had a significant impact on early

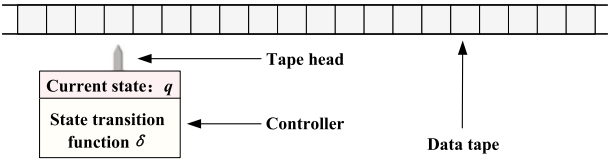


Fig. 2.19 Structure of the turing machine

machine learning and computational biology. In addition, Turing wrote a paper on the chemical basis of morphogenesis and predicted oscillating chemical reactions. The Turing test he proposed is still considered the standard for judging machine intelligence.

The following introduces various types of Turing machines, including deterministic and non-deterministic Turing machines and their variants, and it can be seen that they are theoretically equivalent, that is, they have Turing equivalence.

2.2.2 Turing Machine

So far, electronic computers cannot effectively solve **NP**-complete problems, the reason is: the computational model of electronic computers is **Turing Machine**, denoted as **M**. The Turing machine is an abstract machine, consisting of an infinitely long **data tape**, a **read-write head**, and a **controller**, as shown in Fig. 2.19. The **data tape** can be infinitely extended in both directions and is divided into small squares, each square storing a symbol from a finite alphabet (including blank symbol □); The **controller** controls the read-write head to "move" according to the set state transition function (**program**), the **read-write head** is always in a current state. "Move" is also known as **Turing operation**, which includes the following four actions:

(1) Read the symbol in the square pointed to by the read-write head;
(2) Erase the symbol in the square pointed to by the read-write head and write a new symbol, but the new symbol can be the same as the erased symbol;
(3) The read-write head moves one square to the left or right, or does not move;
(4) Change the state of the controller, but the new state can be the same as the state before the move.

Actions (2)–(4) are based on action (1), the current state of the read-write head, and the transition function δ. If δ is single-valued, then **M is deterministic**; if δ is multi-valued, then **M is non-deterministic**. This article only considers deterministic Turing machines. Specifically, a Turing machine can be represented as the following five-tuple

- $(Q, \Sigma, \Gamma, \square, \delta)$ is a finite state set, including the initial state q_0 and the acceptance state set H;

- Σ is the alphabet, each Turing machine **M** has a specified input alphabet Σ,
- Γ is the finite alphabet available on the data tape, where $\Sigma \subseteq \Gamma$,
- $\square$ is the blank symbol, where $\square \in \Gamma$ and $\square \notin \Sigma$,
- δ is the transition function (also known as the program).

Here, the transition function δ is described in the context of a Turing machine **M**. It specifies the actions to be taken based on the current symbol read from the tape and the current state of the machine. These actions include erasing the symbol, writing a new symbol, moving the machine's head in a certain direction (left or right), and transitioning to a new state. Specifically, $\delta : (Q - H) \times \Gamma \mapsto Q \times \Gamma \times \{R, L, S\}$, where R and L represent the read-write head moving right and left, respectively, and S represents no movement. For the case where the read-write head does not move, it can be replaced by moving left and right once each, so the transition function can also be defined as $\delta : (Q - H) \times \Gamma \mapsto Q \times \Gamma \times \{R, L\}$.

The Turing machine **M** is in a state $q \in Q$ at any time, **state** represents the internal information of the Turing machine at a specific moment. The **initial state** q_0 is the state of the Turing machine at the start of execution; if the transition function stops executing, that is, the current state and the character read by the read-write head are not defined in δ, the Turing machine halts; if it halts in an **acceptance state**, the computation is successful; otherwise, the computation fails.

Initially, a finite string on Σ (as input) is written in adjacent squares on the tape, and the rest of the squares on the tape are empty, the read-write head scans the leftmost symbol of the input, at this time **M** is in the initial state q_0, and in each subsequent step, **M** is in the state $q \in Q$. And the read-write head of **M** reads the symbols on the small squares on the subband $x \in \Gamma$, then based on δ, it performs the corresponding action of $(q.x)$, including scanning the symbols on the square and deleting x, writing new symbols, then moving the read-write head left or right, and obtaining a new state. Here is a simple example to illustrate the operation mechanism of the Turing machine. Define Turing machine **M**, its input alphabet $\Sigma = \{1\}$, finite state set $Q = \{q_1, q_2, q_3\}$ (where $q_1 \triangleq q_0$), the input finite string is 111$\square$11, the state transition function δ is defined as follows:

That is, $\delta(q_1, 1) = q_1 1R$, $\delta(q_1, \square) = q_2 1R$, $\delta(q_2, 1) = q_2 1R$, $\delta(q_2, \square) = q_3 1L$, $\delta(q_3, 1) = q_3 \square S$, $\delta(q_3, \square) = q_3 \square S$, where $\delta(q_i, x_i) = \delta(q_j, x_j, k)$, $(x_i, x_j \in \{1, \square\}$ and $k \in \{R, L, S\})$, indicates that when the current state

δ	$\square$	**1**
q_1	$q_2$1R	$q_1$1R
q_2	$q_3\square$L	$q_2$1R
q_3	$q_3\square$S	$q_3\square$S

is q_i and the read character is x_i, the action is to change the current state of **M** to q_j, write character x_j in the current square, move the read-write head to the right $(k = R)$, move to the left $(k = L)$, or remain unchanged $(k = S)$. The position of the initial state q_1 and the specific operation process are shown in Fig. 2.20. If the number of consecutive 1s represents a specific number, then the above process

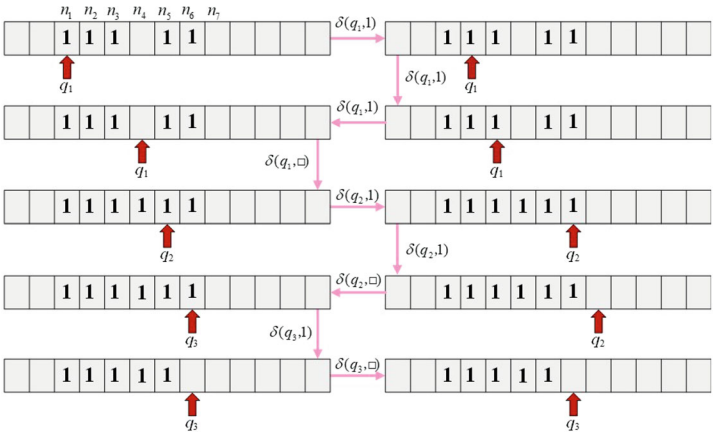


Fig. 2.20 An example of addition operation $3 + 2 = 5$

can implement the addition of the numbers 3 and 2. Therefore, the defined Turing machine **M** can implement the addition of any two numbers.

After the concept of the Turing machine was formally proposed, many variants of the Turing machine were produced, such as multi-tape Turing machines and probabilistic Turing machines, etc. Here we will introduce the multi-tape Turing machine in detail. According to the number of tapes read by the Turing machine, it can be divided into single-tape Turing machines and multi-tape Turing machines. The previously discussed Turing machine is assumed to be a single-tape Turing machine. Multi-tape Turing machines use multiple tapes, each with its own read-write head. Note that any multi-tape Turing machine, no matter how many tapes it has, can be simulated by a single-tape Turing machine. Therefore, it can be considered that multi-tape Turing machines and single-tape Turing machines have the same computational efficiency. The Turing machine is a simple computational model. Many more complex computers seem to be much stronger than Turing machines. However, the Church-Turing thesis expresses a general consensus: any function that can be calculated by a reasonable computational model can be calculated by a Turing machine. That is to say, these computers do not have stronger computational capabilities compared to Turing machines. Turing equivalence refers to the concept in computational theory that two computational models (usually two Turing-complete models) can simulate and be equivalent to each other. Specifically, if one computational model can simulate all the computational processes of another computational model, and vice versa, then these two models are Turing equivalent. The concept of Turing equivalence originates from the theory of Turing machines. The Turing machine is an abstract mathematical model that can describe a computer with infinite storage capacity. Turing proved that as long as a computational model can achieve the same computational power as a Turing machine, that is, it can calculate all computable functions, then this model is Turing equivalent. Most computational models are Turing equivalent, such as λ calculus and general

recursive functions. Turing equivalence means that although they may differ in form and operation, they can calculate the same set of computable functions, so theoretically they are equivalent.

2.2.3 Computability

The theory of computability originated from the work of Turing, Church, and Gödel in the 1930s. The early research on **P** class and **NP** class in terms of computability is deterministic language class and computable enumeration language class respectively. Let **L** be a language, then L is **determinable** if and only if there exists a Turing machine **M** that satisfies $L = L(\mathbf{M})$ and **M** halts for all inputs; **L** is **semi-decidable** if and only if there exists a Turing machine M such that $L = L(\mathbf{M})$, that is, there exists a computable "check relation" $R(x, y)$ such that $L = \{x | \exists y R(x, y)\}$.

Next, we explore computability from the perspective of functions, that is, viewing languages as functions from Σ^* to Σ. In this way, not all functions are computable. The following theorem demonstrates the existence of non-computable functions. In fact, this theorem proves the existence of non-computable functions with a value range of $\{0, 1\}$. Non-computable functions with a value range of $\{0, 1\}$ are also **undecidable languages**.

Theorem 2.12 ([23]) *There exists a function UC:* $\{0, 1\}^* \to \{0, 1\}$ *that cannot be computed by any Turing machine.*

The Halting Problem The function HALT takes an ordered pair as input, and it outputs 1 if and only if the represented Turing machine will halt on the input within a finite number of steps [23]. HALT is a function that needs to be computed. Because, after given a computer program and input, we definitely want to know whether the program will fall into an infinite loop on this input. If the computer can compute the HALT function, then designing bug-free computer software and hardware will become much easier. Unfortunately, it has been proven that the computer cannot compute this function, even if the running time is allowed to be arbitrarily long.

Theorem 2.13 ([23]) *The function HALT cannot be computed by any Turing machine.* □

Theorem 2.13 shows that the halting problem is undecidable. However, the halting problem is semi-decidable.

Next, we introduce a basic concept in the theory of computability—**reducibility**.

Definition 2.1 Let L_1 and L_2 be two languages on symbol sets Σ_1 and Σ_2 respectively, that is, $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$. Then, L_1 is **many-to-one reducible to** L_2 (or L_1 is **polynomial time reducible to** L_2) if and only if there exists a computable function f (that is, there exists a deterministic Turing machine **M** such that for any $x \in \Sigma_1^*$ as input, **M** can halt in polynomial time and output $f(x)$:

$\Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in \Sigma_2$). Use $L_1 \preceq L_2$ to denote that L_1 is **many-to-one reducible to** L_2. The computable function that satisfies the definition is called a L_1 **to** L_2 **polynomial time transformation**.

If $L_1 \preceq L_2$ and L_2 is decidable, then it is easy to infer that L_1 is also decidable. This discovery can be used to illustrate the undecidability of languages. For example, if the halting problem can be reduced to a language L in a many-to-one manner, then L is undecidable. Many-to-one reduction has transitivity.

Theorem 2.14 ([23]) *Suppose L_1, L_2, L_3 are languages on the symbol sets Σ_1, Σ_2, Σ_3 respectively, and $L_1 \preceq L_2$ and $L_2 \preceq L_3$, then $L_1 \preceq L_3$.*

2.2.4 Computational Complexity

Computational complexity is a field in computer science that studies the inherent difficulty of problems and the resources required to solve these problems. Based on this, problems can be divided into several classes, such as **P** class, **NP** class, **coNP** class, etc. Below is a detailed introduction to them.

2.2.4.1 P Class and NP Class

The concepts of **P** class and **NP** class were proposed in the 1960s. Simply put, they provide a basis for measuring the difficulty of problems. The **P** class is a set of problems that a deterministic Turing machine can solve in polynomial time, where polynomial time refers to the transition function of the deterministic Turing machine executing at most polynomial times. Similarly, the **NP** class is also a set of problems that a non-deterministic Turing machine can solve in polynomial time.

(1) Description of P/NP Problem
In general, **P** class and **NP** class problems are decision problems, that is, problems that only require a "yes" or "no" answer. Specifically, decision problems can be described in language. Let Σ be a finite alphabet with at least two elements, and let Σ^* be the set of all finite strings over Σ. Then, a language over Σ is a subset L of Σ^*. For a Turing machine M with an input symbol set Γ and an input w, if **M** halts in an accepting state, then **M** is said to **accept** w; otherwise, M is said to **reject** w. Let $L(M)$ denote the set of all languages by **M**:

$$L(M) = \{w \in \Sigma^* | \mathbf{M} \text{ accepts } w\}$$

For an input w, let $t_M(w)$ denote the number of times the transition function is executed by machine **M** when it halts. If **M** never halts, then $t_M(w)$ is undefined (or can be considered as ∞ if we are discussing the theoretical limit). For any natural number n, let $T_M(n)$ represent the maximum number of times the transition function

is executed by **M** when it halts for all n:

$$T_M(n) = \max\{t_M(w)|w \in \Sigma^n\},$$

where Σ^n denote the set of all strings of length n over the alphabet Σ. If there exists a constant k such that $T_M(n) \leq n^k + k$ for all n, then the running time of machine **M** is said to be **polynomial**. Based on this, the class P is defined as the set of languages accepted by Turing machines that operate in polynomial time:

$$\mathbf{P} = \{L|L = L(\mathbf{M}), \mathbf{M} \text{ is a Turing machine with polynomial running time}\}.$$

NP stands for "Nondeterministic Polynomial time". To provide a formal definition, we first introduce the concept of a "**checking relation**". A checking relation is a binary relation $R \subseteq \Sigma^* \times \Sigma_1^*$, where Σ^* and Σ_1^* are two finite sets of symbols. Each checking relation R is associated with a language L_R on $\Sigma^* \cup \Sigma_1^* \cup \{\#\}$, where $\# \notin \Sigma$ and $L_R = \{w\#y \mid R(w, y)\}$, then R is said to be **polynomial** time if and only if $L_R \in P$. Based on this, a formal definition of the NP class can be given:

A language L on Σ belongs to the NP class if and only if there exists a constant k and a polynomial time checking relation R such that for all $w \in \Sigma^*$,

$$w \in L \Leftrightarrow \exists y(|y| < |w|^k \text{ and } R(w, y)),$$

where $|w|$ and $|y|$ represent the lengths of w and y respectively.

Question 2.1 Is P=NP?

It is easy to see that the answer to Question 2.1 is not limited by the size of the symbol set Σ (assuming $|\Sigma| \geq 2$), this is because any given size of symbol set can be encoded into a binary symbol set.

P $\subseteq$ **NP** is trivial, this is because: for a language L on the symbol set Σ, if $L \in$ **P**, then we can define a polynomial time checking relation $R \subseteq \Sigma^* \times \Sigma^*$ as follows: for any $w, y \in \Sigma^*$, $R(w, y)$ is true if and only if $w \in L$.

Question 2.1 has always been an intricate open question and remains a focus of research at the time of writing this book.

(2) NP-complete Problems

If a problem belongs to **NP**, and all problems in **NP** can be reduced to it in polynomial time, then the problem is called **NP**-complete. The **NP**-complete class refers to the set of all **NP-complete** problems. The concept of the **NP**-complete class was proposed by scientists in North America and the Soviet Union in the late 1960s and early 1970s. It is a subset of the **NP** class, and these problems have significant research significance. The proposal of the **NP**-complete class deepened people's understanding of **P** and **NP** problems, because if it is proven that an **NP**-complete problem belongs to the **P** class, then it also proves that **P=NP**.

Polynomial time computation was first proposed in the 1960s by Cobham [24] and Edmonds [25]. Edmonds [25] referred to polynomial time algorithms as "good

algorithms" and linked them to easy-to-handle algorithms. In 1971, Cook [26] introduced the concept of **NP**-completeness and proved that several problems are **NP**-complete, including 3-SAT and subgraph isomorphism problems, etc. These results were subsequently utilized by Karp [27], who proved that 21 problems are **NP**-complete. Karp proposed the standard concepts of **P** and **NP**, and redefined **NP**-completeness through polynomial time many-to-one reduction, where polynomial time algorithms refer to algorithms with a running time at most n^k for inputs of size n, where k is a constant. Since then, research on computational complexity has received widespread attention.

The standard definition of **NP**-complete ness is very similar to the above definition.

Definition 2.2 Let L_1 and L_2 be two languages on symbol sets Σ_1 and Σ_2 respectively, i.e., $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$. Then, L_1 is **p-reducible** to L_2, denoted as $L_1 \preceq_p L_2$, if and only if there exists a polynomial time computable function $f : \Sigma_1^* \to \Sigma_2^*$ such that for all $x \in \Sigma_1^*$, $x \in L_1$ if and only if $f(x) \in L_2$.

Definition 2.3 A language L is **NP**-complete if and only if L belongs to NP and for every language L' that belongs to **NP**, the polynomial-time reducibility $L' \preceq_p L$ holds.

It is easy to prove that $\preceq_p$ is also transitive, hence the following conclusion.

Theorem 2.15

(1) If $L_1 \preceq_p L_2$ and $L_2 \in \mathbf{P}$, then $L_1 \in \mathbf{P}$;
*(2) If L_1 is **NP**-complete, $L_2 \in \mathbf{NP}$ and $L_2 \preceq_p L_1$, then L_2 is **NP**-complete.*
*(3) If L is **NP**-complete and $L \in \mathbf{P}$, then **P = NP**.*

□

Theorem 2.15(2) provides a fundamental method for proving that a new problem is **NP**-complete, and also illustrates that any two **NP**-complete problems can be transformed into each other in polynomial time. Therefore, if one **NP**-complete problem can be solved in polynomial time, then all other problems can also be solved in polynomial time [which is also the implication of Theorem 2.15(1)]. Theorem 2.15 (3) suggests that the search for a polynomial-time algorithm for **NP**-complete problems is likely futile.

Roughly speaking, **P** class problems are those that can be solved in polynomial time, that is, problems that can be solved within $O(n^k)$, where k is a constant, and n is the size of the problem input. Problems in the **NP**-class are "verifiable" in polynomial time, that is, given a solution to a problem, it can be verified whether it is correct within polynomial time of the problem input size. In other words, for **NP** problems, there may not be a known fast method to get the answer to the problem, but if a candidate answer is given, it can be verified in polynomial time whether the answer is a solution to the known problem. Therefore, any problem in the **P** class belongs to the **NP** class, because any problem can be solved in polynomial time, then given a solution to a problem, it can definitely be verified for its correctness in polynomial time. **NP**-complete problems are a subset of **NP** problems, and are

the most difficult problems in the **NP** class. There is also a class called **NP**-hard problems, which are problems that **NP** problems can be p-reduced to. In a word, their characteristic is "problems that are at least as hard as **NP** problems", that is, **NP**-hard problems are at least as hard as **NP** problems. Obviously, **NP**-hard problems may belong to **NP** or may not belong to **NP**, they may be undecidable problems. For more introductions to complexity theory, please refer to the works of Paradimitriou [28] and Sipser [29].

(3) SAT Problem

Theorem 2.15 (2) provides a method for proving **NP**-complete problems, however, to prove a new problem is **NP**-complete, it still needs to rely on a known **NP**-complete problem. This raises a question: how was the first **NP**-complete problem obtained? In fact, the first **NP**-complete problem was not obtained through the above method, but through theories related to Turing machines. Cook [26] first obtained such a problem in 1971, namely the satisfiability problem (SAT problem), which is introduced below.

Members of the **NP** problem can be represented as decision problems (questions that only need to answer yes or no), and the corresponding language can be understood as a collection of some strings, where these strings encode "YES" instances into decision problems through standard encoding methods. Based on this, the **SAT problem** can be described as follows:

Problem 2.1 (SAT Problem) Given a propositional formula F composed of conjunction ($\wedge$), disjunction ($\vee$), and negation ($\neg$), determine whether F is satisfiable.

The SAT problem is a type of constraint satisfaction problem, its decision problem is whether there exists a set of variable assignments that make the proposition true. In a Boolean formula, it contains such components: Boolean variables (taking values of 0 or 1), Boolean connectives (conjunction, disjunction, negation), parentheses. For a Boolean formula, if there exists some assignment of 0 or 1 to its variables that makes the truth value of the formula 1, then it is called **satisfiable**. Based on this, an instance of the SAT problem is a Boolean formula φ composed of the following components:

(1) n Boolean variables, $x_1, x_2, \ldots, x_n$.
(2) m Boolean connectives. The Boolean connectives include $\wedge$ (AND), $\vee$ (OR), and $\neg$ (NOT).
(3) Parentheses. It is assumed that there are no redundant parentheses, meaning each Boolean connective has at most one pair of parentheses.

For example, for the formula $(x_1 \vee \neg x_1 \vee x_3) \wedge (\neg x_1 \vee x_3) \wedge (x_2 \vee \neg x_3)$, when its variable truth value assignment is $x_1 = 0$, $x_2 = 1$ and $x_3 = 1$, then the truth value of this formula is 1, so it is satisfiable. But not all Boolean formulas are satisfiable; for example, $x \wedge \neg x_1$ is unsatisfiable. Note that all **NP** problems can be transformed into SAT problems.

Since for a given Boolean formula and any assignment of variables, it can be verified in polynomial time whether the Boolean formula is true, the SAT problem

belongs to the **NP** class. Specifically, a polynomial-time checking relation $R(x, y)$ is defined, where x and y are related by R if and only if x encodes a propositional formula F and y encodes a set of truth assignments to the variables in F that make F satisfiable. Cook [26] proved in 1971 that the SAT problem is **NP**-complete (Levin [30] also proved this in 1973). Cook's method involved demonstrating that for every polynomial-time Turing machine M, if M can recognize the checking relation $R(x, y)$ of an NP language L, then there exists a polynomial-time algorithm A such that: A accepts a string x as input and generates a corresponding propositional formula F_x where F_x is satisfiable if and only if M accepts (x, y) for some string y of length less than or equal to $|x|^{O(1)}$. For the detailed proof, one can refer to the literature [31].

Theorem 2.16 *The SAT problem is **NP**-complete.* □

Based on Theorem 2.16, a new problem may be proven to be **NP**-complete using **p-reduction** from the SAT problem. In addition, if $B \in NP$ cannot be proven, then we say the corresponding problem B is **NP**-hard.

From Theorem 2.16 and the transitivity of p-reduction, all **NP** problems can be reduced to SAT problems. That is to say, if the SAT problem is solved, all NP problems are solved. This also shows from another aspect that all **NP**-complete problems are polynomial time equivalent.

An important special case of the SAT problem is 3-SAT, which is introduced below.

Problem 2.2 (3-SAT Problem) Let $X = \{x_1, x_2, \ldots, x_n\}$ be a finite set of Boolean variables, where each $x_i \in \{0, 1\}$. Let $C = C_1 \wedge C_2 \wedge \cdots \wedge C_m$ be a conjunctive normal form, where each C_i is a disjunction of three variables. The question is whether there exists a truth assignment to the variables in X such that C is true, i.e., each C_i is true.

To illustrate the *p-reduction* strategy, a detailed proof that the 3-SAT problem is **NP**-complete is presented below. This proof originates from Cook [26].

Theorem 2.17 *The 3-SAT problem is NP-complete.*

Proof Firstly, since the SAT problem belongs to the **NP** class, the 3-SAT problem also belongs to the **NP** class. Next, we prove SAT $\preceq_p$ 3-SAT. Let ϕ be a Boolean formula in conjunctive normal form. In polynomial time, construct a new Boolean formula ψ in conjunctive normal form such that:

- Each clause of ψ contains 3 variables.
- ϕ is satisfiable if and only if ψ is satisfiable.

If a clause in ϕ contains only 1 variable, say x, then replace this clause with 4 clauses, each containing 3 variables, where y and z are two new variables as follows.

$$x = (x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z) \wedge (x \vee \neg y \vee \neg z).$$

If a clause in ϕ contains only 2 variables, say $x_1 \vee x_2$, then replace this clause with 2 clauses, each containing 3 variables, where w is a new variable. That is,

$$(x_1 \vee x_2) = (x_1 \vee x_2 \vee w) \wedge (x_1 \vee x_2 \vee \neg w)$$

Now consider a clause in ϕ containing k variables $x_1, x_2, \cdots, x_k$, where $k \geq 4$. In this case, add $k-3$ new variables $y_1, y_2, \cdots, y_{k-3}$ and construct the following $k-2$ clauses, each containing 3 variables:

$$(x_1 \vee x_2 \vee y_1), (\neg y_1 \vee x_3 \vee y_2), \ldots, (\neg y_{k-4} \vee x_{k-2} \vee y_{k-3}), (\neg y_{k-3} \vee x_{k-1} \vee x_k).$$

It is easy to verify that the conjunction of these clauses is equivalent to the original clause. Therefore, ϕ is satisfiable if and only if ψ is satisfiable, which proves SAT $\preceq_p$ 3-SAT. □

As research continues, an increasing number of **NP**-complete problems have been identified, such as the Subset Sum Problem (whether there exists a subset in a given set of positive integers whose sum equals a predetermined target value), and graph problems [for example, given a graph G, does G contain a Hamiltonian Cycle? Does G have a 3-coloring? Does G have an independent set of k vertices?] Reference [31] provides a detailed introduction to the study of **NP**-complete problems and lists 300 **NP**-complete problems. According to incomplete statistics, there are at least several thousand NP-complete problems discovered to date.

There are also some interesting problems within the **NP** class (such as the graph isomorphism problem, which determines whether two given undirected graphs are isomorphic). As of the date of publication of this book, it is still unknown whether they belong to the **P** class or are **NP**-complete.

On May 24, 2000, the Clay Mathematics Institute (CMI) announced seven mathematical problems at a conference held at the École Polytechnique in Paris (also known as the Paris Millennium Meeting), with a total reward of $7 million for their solutions. Hence they are referred to as the Millennium Prize Problems [32]. According to their rules, solving each problem comes with a reward of $1 million [33]. The P/NP problem is the first of these seven problems, highlighting its extraordinary significance. The problem aims to determine whether every language that can be accepted by a non-deterministic algorithm in polynomial time can also be accepted by some deterministic algorithm in polynomial time. Here, "polynomial time" refers to the upper bound of the algorithm's running time being a polynomial function of the input size.

Regarding this question, a survey of one hundred experts in mathematics and computer science was published in 2001, with 61 respondents providing a negative answer [34]. In 2012, William [35] conducted the survey again, and 84 individuals gave a negative response, suggesting that P $\neq$ NP. As for the P/NP problem, as early as 1970, Cook and Levin proved that if there exists an NP-complete problem that is solvable in polynomial time, then all NP problems are solvable in polynomial time [36]. Therefore, it is only necessary to study one such problem. To precisely

describe the P/NP problem, one must rely on a formal model of computation. In computational theory, the standard model of computation is the Turing machine [21]. Although the Turing machine was proposed before the physical computer was constructed, it has always been considered an appropriate model for defining computable functions.

2.2.4.2 coNP Problem

The **coNP** class is also a collection of problems. As its name suggests, the **coNP** class is related to the **NP** class and can be defined in terms of NP. Here is an alternative definition of **coNP**. For a language $L \subseteq \Sigma*$, if there exists a polynomial function $f : \mathbb{N} \to \mathbb{N}$ and a polynomial-time Turing machine **M**, such that for any $x \in \Sigma^*$,

$$x \in L \Leftrightarrow \forall u \in \Sigma^{f(x)}, M(x, u) = 1,$$

then L is in coNP, where $M(x, u) = 1$ indicates that the Turing machine M accepts the pair (x, u).

coNP and **coNP**-complete relationship is similar to **NP** and **NP**-complete relationship. Simply put, **coNP**-complete $\subseteq$ **coNP**, and **coNP**-complete is composed of the most difficult problems in **coNP**. That is to say, all problems in **coNP** can be polynomially reduced to problems in **coNP**-complete. It is currently unknown whether **coNP** and **NP** are equal. However, under the assumption that **coNP** $\neq$ **NP**, it can be proved that: **NP**-hard and **coNP** do not intersect. Note that if **coNP** $\neq$ **NP**, then **NP**-complete $\cap$ **coNP** $= \emptyset$ and **NP** $\cap$ **coNP**-complete $= \emptyset$ [37].

Theorem 2.18 *If* **coNP** $\neq$ **NP**, *then* **NP**-$hard \cap$ **coNP** $= \emptyset$.

Proof Assume there exists a language $L \in$ **NP**-$hard \cap$ **coNP**. According to the definition of **coNP**, there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial time Turing machine **M**, such that for any $x \in \Sigma^*$ we have $x \in L$ if and only if for any $u \in \Sigma^{p(x)}$, $M(x, u) = 1$. Also according to the definition of NP-hard, there exists a language $L' \in NP\text{-}complete$ such that $L' \preceq_p L$. Therefore, there exists a polynomial time computable function $f : \Sigma \to \Sigma$ such that for any $x \in \Sigma^*$ we have $x \; in L' \Leftrightarrow f(x) \in L$. Combining the above two definitions, for any $x \in \Sigma^*$ we have $x \in \Sigma^* \Leftrightarrow \forall u \in \Sigma^{p(x)}$, $M(f(x), u) = 1$. Note that the composite operation of polynomials satisfies closure. Therefore, the definition of **coNP**, $L' \in$ **coNP** contradicts with **NP** $- complete \cap$ **coNP** $= \emptyset$. □

For **coNP**, the following theorem is obvious.

Theorem 2.19 **P** $\subseteq$ **NP** $\cap$ **coNP**.

So far, we have introduced the **P** class, N**P** class, **NP**-hard class, **NP**-complete class, **coNP** class, and **coNP**-complete class. Figure 2.21 shows their relationships.

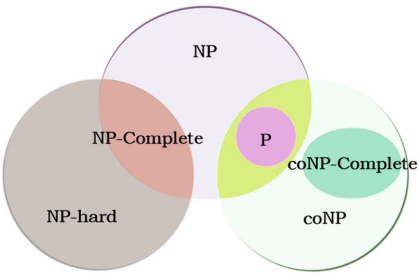


Fig. 2.21 The relationship between different classes of problems

References

1. Xu, J.: The Theory of Maximal Planar Graphs (Vol. 1: Structure-Construction-Coloring). Science Press, Beijing (2019).
2. Bang, J., Gutin, G.: Theory, Algorithms, and Applications of Directed Graphs. Translated by Yao, B., Zhang, Z.F. Science Press, Beijing (2009).
3. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. Springer, New York (2008).
4. Lovász, L.: On the Shannon capacity of a graph. IEEE Trans. Inf. Theory **25**(1), 1–7 (1979).
5. Babai, L.: Graph isomorphism in quasipolynomial time. In: Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing, pp. 684–697 (2016).
6. Grohe, M., Schweitzer, P.: The graph isomorphism problem. Commun. ACM **63**(11), 128–134 (2020).
7. McKay, B.D., Min, Z.K.: The value of the Ramsey number R(3, 8). J. Graph Theory **16**(1), 99–105 (1992).
8. McKay, B.D., Radziszowski, S.P.: R(4, 5) = 25. J. Graph Theory **19**(3), 309–322 (1995).
9. Brooks, R.L.: On colouring the nodes of a network. Proc. Cambridge Philos. Soc. **37**, 194–197 (1941).
10. Vizing, V.G.: On an estimate of the chromatic class of a p-graph (Russian). Diskret. Analiz. **3**, 25–30 (1964).
11. Gupta, R.P.: The chromatic index and the degree of a graph. Notices Amer. Math. Soc. **13**, abstract 66T-429 (1966).
12. Behzad, M.: Graphs and their chromatic numbers. Ph.D. dissertation, Michigan State University, Michigan, United States (1965).
13. Vizing, V.G.: Some unsolved problems in graph theory. Russ. Math. Surv. **23**, 125–142 (1968).
14. Zykov, A.A.: On some properties of linear complexes (Russian). Math. Sbornik **24**, 163–188 (1949).
15. Boppana, R.: Approximating maximum independent sets by excluding subgraphs. Springer Berlin Heidelberg **32**(2), 13–25 (1990).
16. Welsh, D.J.A., Powell, M.B.: An upper bound on the chromatic number of a graph and its application to timetabling problems. Comput. J. **10**, 85–87 (1967).
17. Kokosinski, Z., Kwarciany, K., Kolodziej, M.: Efficient graph coloring with parallel genetic algorithms. Comput. Informatics **24**, 123–147 (2005).
18. Xu, J., Qiang, X., Zhang, K., et al.: A parallel type of DNA computing model for graph vertex coloring problem. In: Proceedings of the IEEE Fifth International Conference on Bio-inspired Computing: Theories and Applications, pp. 231–235 (2010).
19. Dempster, M.A.H.: Two algorithms for the time-table problem. In: Welsh, D.J.A. (ed.) Combinatorial Mathematics and its Applications, pp. 63–65. Academic Press, New York (1971).
20. de Werra, D.: On some combinatorial problems arising in scheduling. Canad. Oper. Res. Soc. J. **8**, 165–175 (1970).

21. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proc. Lond. Math. Soc. **2**(42), 230–265 (1936).
22. Turing, A.M.: Systems of logic based on ordinals. Proc. Lond. Math. Soc. Ser. 2 **45**, 161–228 (1939).
23. Sanjeev, A., Barak, B.: Computational Complexity: A Modern Approach. Cambridge University Press, Cambridgeshire, England (2009).
24. Cobham, A.: The intrinsic computational difficulty of functions. In: Bar-Hille, Y. (ed.) Proceedings of the 1964 International Congress for Logic, Methodology, and Philosophy of Science, pp. 24–30. Elsevier/North-Holland, Amsterdam (1964).
25. Edmonds, J.: Minimum partition of a matroid into independent subsets. J. Res. Nat. Bur. Standards Sect. B **69**, 67–72 (1965).
26. Cook, S.: The complexity of theorem-proving procedures. In: ACM Symposium on Theory of Computing, pp. 151–158. ACM, New York (1971).
27. Karp, R.M.: Reducibility among combinatorial problems. In: Miller, R.E., Thatcher, J.W. (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press, New York (1972).
28. Papadimitriou, C.: Computational Complexity. Addison-Wesley, Boston (1994).
29. Sipser, M.: Introduction to the Theory of Computation. PWS Publ., Boston (1997).
30. Levin, L.: Universal search problems (in Russian). Problemy Peredachi Informatsii **9**, 265–266 (1973).
31. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman and Co., San Francisco (1979).
32. Jaffe, A.M.: The Millennium Grand Challenge in Mathematics. Notices of the AMS, 652 (2000).
33. Carlson, J.A., Jaffe, A., Wiles, A.: The Millennium Prize Problems. American Mathematical Society and Clay Mathematics Institute, Providence, RI (2006).
34. Gasarch, W.: The P=?NP poll. ACM SIGACT News **33**(2), 34–47 (2002).
35. Gasarch, W.: Guest Column: The second P=?NP poll. ACM SIGACT News **43**(2), 53–77 (2012).
36. Siper, M.: Introduction to the Theory of Computation. Translated by Tang, C.J., Chen, P., Xiang, Y., Liu, Q.H. Mechanical Industry Press, Beijing (2000).
37. Hartmanis, J., Immerman, N.: On complete problems for NP ∩ coNP. In: Proceedings of the 12th Colloquium on Automata, Languages and Programming, pp. 250–259. Springer Berlin Heidelberg, Nafplion, Greece (1985).

Chapter 3
Biological Computing: Data

As known from Chap. 1, biological computing is divided into DNA computing, RNA computing, and protein computing. The data used are DNA, RNA, and proteins, respectively. This chapter mainly introduces the physical and chemical characteristics and computational characteristics of these three types of biological macromolecules, which are the basis of biological computing.

3.1 DNA Molecules

DNA molecules generally exist in the form of high-molecular-weight double strands. Changes in the microenvironment and molecular composition can also form single-strand, triple-strand, quadruple-strand DNA molecules, and even DNA\RNA hybrid molecules [1]. Each single-strand DNA is composed of a sequence of deoxyribonucleotides, each of which can be represented by its alkaline molecule. Therefore, the sequence of deoxyribonucleotides is composed of repeatable characters A,T,G,C. The connection method of adjacent deoxyribonucleotides is a phosphodiester covalent bond. DNA single-strand molecules form DNA double strands through hydrogen bond attraction, and double-strand DNA in the body will further twist and compress to form a higher-order structure, used for life genetic information storage and physiological activity regulation (Fig. 3.1).

DNA computing is a new type of computing mode that uses DNA molecules as data and biological enzymes or biochemical operations as information processing "tools" [2]. In the DNA computation model, the data, namely DNA molecules, need to be given a DNA encoding. DNA encoding is not only necessary for DNA computation, but also for the entire genetic engineering and DNA storage. We will discuss this in detail in Chap. 5. The structure and characteristics of DNA molecules are the cornerstone of DNA computation, which will be introduced in detail below, and you can also refer to reference [2].

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_3

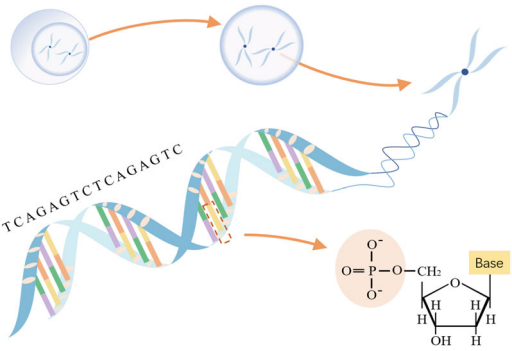


Fig. 3.1 Schematic diagram of DNA molecule composition and structure

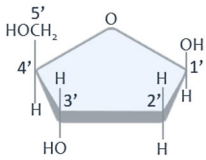


Fig. 3.2 Molecular structure of deoxyribose

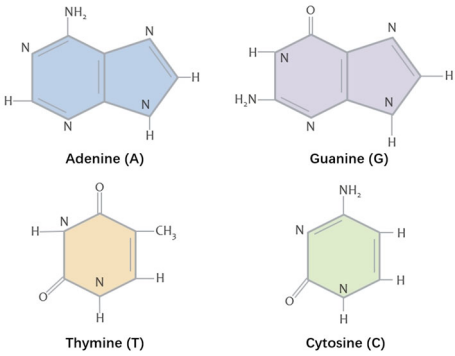


Fig. 3.3 Four nitrogenous bases

3.1.1 Deoxyribonucleic Acid

Deoxyribonucleic acid is covalently condensed from one molecule of phosphoric acid, one molecule of pentose, and one molecule of organic base. The pentose in DNA is 2-deoxyribose, that is, the hydroxyl group on the 2-position of ribose is replaced by hydrogen (see Fig. 3.2).

DNA molecules have two types of nitrogenous bases: purines and pyrimidines. From an organic chemistry perspective, purines have a double-ring molecular structure, while pyrimidines have a single-ring molecular structure. Purines are generally divided into adenine (A) and guanine (G); pyrimidines are divided into thymine (T) and cytosine (C). Their names and molecular structures are shown in Fig. 3.3.

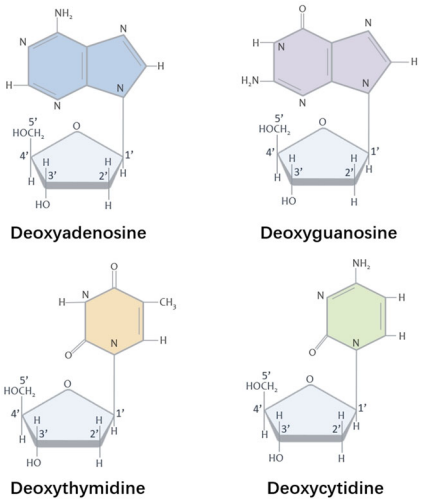


Fig. 3.4 Four deoxyribonucleosides

The 1st C atom on the pentose molecule is connected to the 9th N atom on the purine or the 1st N atom on the pyrimidine by a β type C-N glycosidic bond to form a nucleoside. There are commonly four types of deoxyribonucleosides according to the base difference: deoxyadenosine, deoxyguanosine, deoxythymidine, and deoxycytidine. Figure 3.4 shows the structures of these four deoxyribonucleosides.

In the complex physiological and biochemical environment within the organism, the hydroxyl group on the pentose ring structure undergoes a key chemical modification of phosphorylation. This process is an important step in the formation of nucleotides, which then play an indispensable role in life activities such as DNA replication. From the perspective of organic chemical structure, the nucleoside and the phosphate groups are closely connected by esterification reactions, together building the basic structural framework of deoxyribonucleic acid. Specifically, the combination of pentose and phosphoric acid molecules forms deoxyribonucleic acid, such as the typical representative deoxythymidine monophosphate (dTMP), where the 'd' prefix clearly identifies its deoxy- (deoxy-) characteristic. As shown in Fig. 3.5, although the four different types of deoxyribonucleic acids have different side chain bases, they share a similar core skeleton structure. In this structure, the binding point of the phosphate group and deoxyribose (i.e., 2-deoxyribose) is precisely located on the 3' or 5' carbon atom of the deoxyribose molecule. This specific binding mode not only provides stable chemical support for the linear extension of the DNA chain, but also ensures the stability of the DNA double helix structure and the accurate transmission of genetic information.

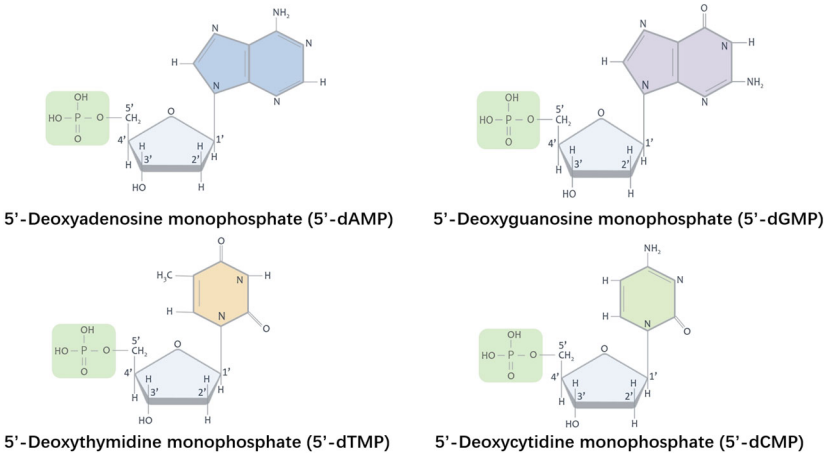


Fig. 3.5 Four types of deoxyribonucleotides

3.1.2 DNA Molecular Structure

A monophosphate nucleotide refers to those nucleotides that only contain one phosphate group. When multiple such nucleotides are sequentially connected by phosphate and pentose (i.e., deoxyribose) to form a long chain, they constitute the basic building blocks of DNA: polynucleotide molecules. The primary structure of DNA describes the arrangement order of these deoxyribonucleotides in the DNA molecule (also often referred to as the base sequence) and the way they are connected by specific chemical bonds. Specifically, the polynucleotide chain of DNA is composed of four different types of deoxyribonucleotides, which may vary in number, and are tightly connected by 3' to 5' phosphodiester bonds. Therefore, the main chain structure of DNA presents a linear long chain feature of alternating phosphate groups and deoxyribose. For linear DNA molecules, it has two free ends: one end is the end of deoxyribose at the 5' position with a hydroxyl (-OH) group, known as the 5' end; the other end is the end of deoxyribose at the 3' position with a hydroxyl group, known as the 3' end. When describing the primary structure of DNA, we usually follow the writing direction from the 5' end to the 3' end, which is intuitively demonstrated in part of Fig. 3.6a.

The secondary structure of DNA molecules is the famous double helix structure model proposed by Watson and Crick in 1953 (as shown in Fig. 3.6b). The core features of this model are summarized as follows:

- Double helix structure: The DNA molecule is composed of two antiparallel polydeoxyribonucleotide chains, which tightly wind around a common central axis in a right-handed helical manner, forming a stable double helix structure.
- Main chain and base position: The backbone of each chain is alternately connected by phosphate groups and deoxyribose, these backbones are located

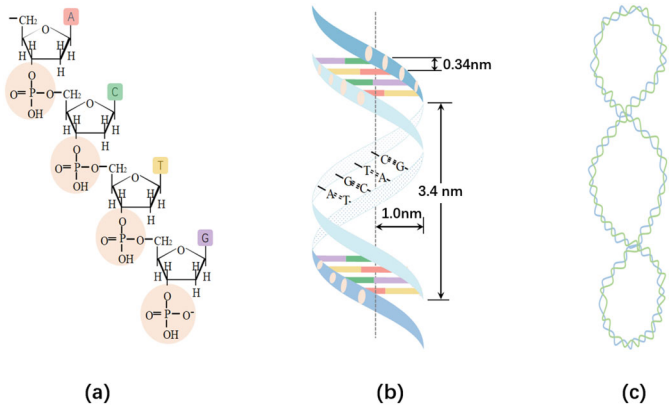


Fig. 3.6 DNA molecular structure

on the outside of the double helix, while the bases are on the inside, arranged perpendicular to the central axis of the helix. The surface of the double helix presents two helical grooves, namely the major groove and the minor groove, which play an important role in the interaction of DNA with other molecules.

- Geometric parameters: The diameter of the double helix is about 2 nm, and each rotation around the central axis contains about 10 base pairs, forming a helical period. The pitch of the entire helix (i.e., the height of the helix for each complete rotation) is 3.4 nm, and the distance between adjacent base pairs is 0.34 nm. These precise dimensions ensure the stability of the DNA double helix structure.
- Base pairing: In the double helix structure, two adjacent bases pair with each other through hydrogen bonds. Specifically, adenine (A) forms two hydrogen bonds with thymine (T), while guanine (G) forms three hydrogen bonds with cytosine (C). This specific pairing method not only maintains the stability of the double helix structure but also ensures the accurate transmission of genetic information.

As for the tertiary structure of DNA, it refers to the complex structure formed by the further spatial folding and twisting of the double-stranded DNA molecule. Among them, the superhelix is an important form of the tertiary structure (as shown in Fig. 3.6c), which further increases the compactness and stability of the DNA molecule and is of great significance for the packaging and storage of DNA in cells.

The importance of base pairing as one of the core characteristics of DNA molecules is self-evident, whether in biological processes in nature, in the application of artificial technologies such as genetic engineering, or in the field of DNA computing deeply explored in this book, this property occupies a fundamental and crucial position. To intuitively demonstrate this principle, we provide a schematic diagram of base pairing in Fig. 3.7. The double-stranded structure of DNA is based on the complementary pairing of bases, which are tightly wound into a unique double helix shape through the formation of hydrogen bonds. Due to the

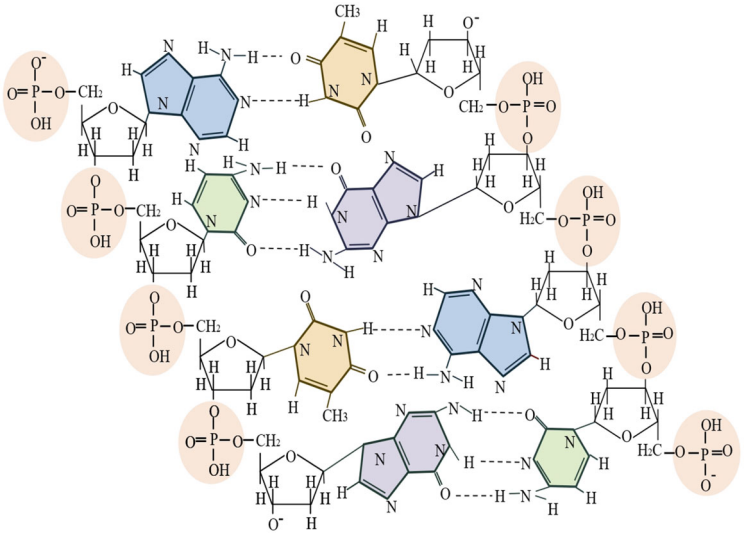


Fig. 3.7 Base pairing in DNA double strands

clever arrangement of geometric configuration and chemical properties, adenine (A) specifically pairs with thymine (T), and guanine (G) matches with cytosine (C). Specifically, A and T can stably form two hydrogen bonds, while G and C can form three more stable hydrogen bonds. This precise pairing method not only ensures the stability of the DNA structure but also makes the paired bases strictly arranged on the same plane, further enhancing the integrity and functionality of the double helix structure.

3.1.3 Types of DNA Molecules

In DNA computing research, we will involve single-stranded DNA molecules, double-stranded DNA molecules, triple-stranded DNA molecules, hairpin DNA molecules, circular DNA molecules, etc. The concept of triple-stranded DNA molecules is given in reference [1], and the rest are introduced separately in this section.

3.1.3.1 Single-Stranded DNA

Single-stranded DNA molecules are essentially a manifestation of the primary structure of DNA, which appears as a linear sequence of nucleotides tightly connected by phosphodiester bonds. In the primary structure of DNA, the combi-

nation mode of phosphate and deoxyribose is constant, while the variable is the base sequence carried by these nucleotides. Therefore, we usually simplify this nucleotide sequence to discuss the base sequence and habitually use A (adenine), C (cytosine), G (guanine), and T (thymine) as representatives to indicate different bases. For example, the base sequence of the DNA sequence in Fig. 3.6a can be intuitively represented by the combination of these letters:

$$5'ACTG3'$$

In the field of DNA computing, single-stranded DNA molecules have been widely regarded as the basic data unit for information processing. For example, in Adleman's pioneering work, he cleverly used single-stranded DNA molecules to represent the vertices of the graph when constructing a DNA computing model to solve the Hamiltonian path in a directed graph. Specifically, he designed a representation method for each edge, which is formed by taking half of the DNA sequences corresponding to the two vertices connected by the edge and concatenating them. Subsequently, using the complementary chains of these edge-corresponding sequences as templates, hybridization reactions are triggered to screen out the required directed paths. This process fully demonstrates the unique advantages and potential of single-stranded DNA molecules as information carriers in DNA computing.

3.1.3.2 Double-Stranded DNA

In the field of DNA computing, some models directly use double-stranded DNA molecules as the basic unit for encoding data. However, fundamentally, whether single-stranded or double-stranded DNA is used for encoding, there is no essential difference in the context of DNA computing. The reasons are mainly two: first, single-stranded DNA molecules often naturally form double-stranded structures through hybridization reactions during the DNA computing process; second, based on the Watson-Crick complementarity principle, once we know the sequence of a single-stranded DNA molecule, we can deduce its corresponding double-stranded DNA molecule structure, and vice versa. Therefore, in the specific practice of DNA computing, we should flexibly choose to use single-stranded or double-stranded DNA molecules according to the actual needs of the problem. This article only proposes a specific marking system for double-stranded DNA molecules for subsequent discussion and analysis.

For a given double-stranded DNA molecule, such as

$$X = \begin{matrix} 5'ACTGTTAAGA3' \\ 3'TGACAATTCT5' \end{matrix}$$

Usually, the corresponding lowercase English letters are used to represent the single-stranded DNA molecule from the 5' end to the 3' end in the double-stranded DNA molecule, that is, the upper half of the double strand, and the lowercase

English letters are used to represent the single-stranded DNA molecule from the 3' end to the 5' end in the double-stranded DNA molecule, that is, the lower half of the double strand. So, for the above example, we have: $x = 5'ACTGTTAAGA3'$ or $\overline{x} = 5'ACTGTTAAGA3'$. Sometimes, when there is no confusion, the 5'-end and 3'-end are omitted. For example, for the above example, we have: $x = ACTGTTAAGA$. We call x the upper chain of X, and $\overline{x}$ the lower chain of X. We call x and $\overline{x}$ complementary chains. The double-stranded DNA molecule X is usually abbreviated as

$$X = \delta(x)$$

That is, the single-stranded DNA molecule x can generate the double-stranded DNA molecule X.

3.1.3.3 Hairpin DNA

Under certain conditions, longer single-stranded DNA molecules can use the attraction of hydrogen bonds to follow the Watson-Crick base pairing rules, that is, A (adenine) and T (thymine), G (guanine) and C (cytosine) form stable hydrogen bond connections. This pairing process promotes some single-stranded DNA molecules to combine with each other through hybridization to form a double-stranded structure, while some single-stranded DNA molecules do not participate in hybridization and maintain their original single-stranded state. Take a specific single-stranded DNA molecule as an example:

$$x = 5'ACTGTTAAGAGGGGATTATTCTTAACAGT3'$$

Clearly, the first 10 bases and the last 10 bases follow the Watson-Crick base pairing principle, forming a complementary correspondence. Under specific conditions, the pairing action between these bases prompts the DNA molecule to fold into a unique structure, as shown in Fig. 3.8. This structure has a stable double-stranded region at one end and a circular structure at the other end. This special DNA molecule configuration is called "hairpin DNA molecule" or simply "hairpin DNA". In the hairpin DNA molecule, we call the tightly paired double-stranded part the "stem", and the circular region composed of unpaired bases is called the "loop".

Hairpin DNA molecules occupy a pivotal position in the field of DNA computing and DNA computer research. Its importance is reflected in several aspects: First,

Fig. 3.8 Hairpin DNA structure

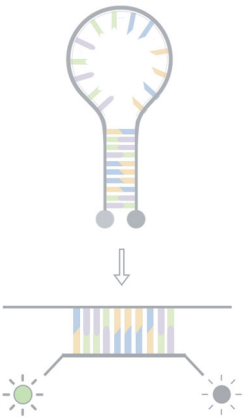


Fig. 3.9 Principle of molecular beacon

Fig. 3.10 Types of DNA molecules with sticky ends

hairpin DNA molecules are cleverly designed and used to make molecular beacons. The basic structure of this beacon is shown in Fig. 3.9. It is mainly used in DNA computing to solve solution detection, providing a powerful tool for verifying experimental results. Secondly, hairpin DNA molecules directly participate in the DNA computing process and become a key component of the DNA computing model for constructing solutions to satisfiability problems and other complex computing tasks. Furthermore, hairpin DNA molecules show great potential in the research of DNA computer models in the field of disease diagnosis and treatment. The stem part can specifically recognize target molecules as diagnostic probes, and the loop part may carry DNA sequences that inhibit disease development, providing new ideas for precision medicine. This pioneering work in this field was first carried out by the research group led by Israeli scholar Ehud Shapiro, and further expanded and deepened in subsequent research. Related results can be seen in reference [3].

3.1.3.4 DNA with Sticky Ends

The DNA molecule structure shown in Fig. 3.10 is DNA with sticky ends. In the practice of DNA computing, this type of DNA molecule data with sticky ends usually needs to be obtained through artificial synthesis or enzyme cutting reactions. As an important carrier of information processing, they have shown a wide range

of application prospects in the field of DNA computing. For example, they have been successfully applied to the paste DNA computing model and the graph vertex coloring DNA computer model, showing their unique advantages in dealing with complex graph theory problems. More broadly speaking, DNA molecules with sticky ends have potential application value in almost all areas of graph theory research. For a deeper understanding of the specific applications and advantages of these molecules in DNA computing, refer to references [4–6], which discuss their in-depth applications and frontier progress in this field.

3.1.3.5 Plasmid DNA

Plasmids are one of the indispensable common carriers in genetic engineering, perfectly meeting a series of conditions required as a genetic engineering carrier [7]. Plasmids are essentially a subcellular level genetic element, neither wrapped in a protein shell nor having an independent life cycle outside the cell. It depends on the host cell for replication and proliferation, stably inherited to the daughter cells with the division of the host cell, but once it leaves the host cell environment, the plasmid cannot survive independently. Plasmids have a certain compensatory effect on the function of the host cell, and they achieve this by regulating various biological processes. Plasmids have the ability to replicate and transcribe autonomously, which is one of the key characteristics of them as gene carriers. They can ensure a constant copy number in daughter cells, thereby stably transmitting and expressing the genetic information they carry. In addition, the existence form of plasmids in cells is flexible and diverse, they can either independently exist in the cytoplasm or integrate into the bacterial chromosomal DNA, this flexibility provides great convenience for genetic engineering operations [7].

Plasmids have been found to be widely present in various organisms, including prokaryotic cells, some eukaryotic cells, Gram-positive and negative bacteria, and specific microorganisms such as E. coli. As stable genetic elements outside the chromosome, plasmids typically range in size from 1 to 200 kilobase pairs (kb), presenting a double-stranded closed circular DNA structure. In terms of structural complexity, plasmids are relatively simple, even more so than viruses. In E. coli, a model organism, scientists have identified and classified various types of plasmids, the most well-known of which include F plasmids, R plasmids, and Col plasmids. F plasmids, also known as F factors or sex factors, have a unique ability to carry genes from the host chromosome and transfer them to recipient cells that originally did not contain the plasmid, achieving horizontal gene transfer. R plasmids, known as resistance factors because they encode one or more antibiotic resistance genes, not only endow bacteria with resistance to specific antibiotics but can also transfer this resistance to recipient cells lacking the corresponding plasmid under suitable conditions, enabling the latter to also acquire antibiotic resistance. As for Col plasmids, they are a type of factor that produces colicins, the genes they encode control the synthesis of colicins. Colicins are a type of protein with strong

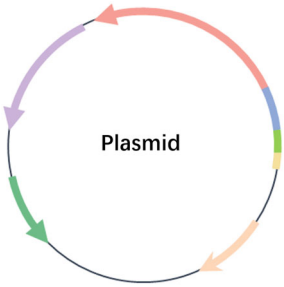


Fig. 3.11 Schematic of plasmid DNA

antibacterial activity, capable of killing bacterial strains that are closely related to the E. coli producing the Col plasmid but do not carry the plasmid.

In the coding design of plasmid DNA, all plasmid DNA molecules suitable for use as gene cloning vectors share three core components: Replicon structure, Selective Marker, and Cloning Site. Specifically, the Replicon structure is the basis for plasmid self-replication, which includes a key replication initiation site (Origin of Replication, abbreviated as ori) responsible for initiating the DNA replication process; it also contains regulatory genes to control the frequency and efficiency of replication, and Replicon coding genes, which are crucial to the plasmid's replication mechanism. The Multiple Cloning Site (MCS) consists of a series of single restriction enzyme cutting sites, providing precise insertion points for exogenous DNA fragments, allowing researchers to insert target genes or DNA sequences into specific locations on the plasmid as needed. Figure 3.11 visually demonstrates the basic structure of a plasmid DNA molecule.

In the manipulation of plasmid DNA molecules, especially for the insertion and deletion of exogenous nucleotide sequences, we mainly rely on Type II restriction endonucleases. Type II restriction endonucleases are composed of a single polypeptide chain and often exist in the form of homodimers in organisms. Its unique properties include the following: First, it can recognize specific nucleotide sequences on the double strands of DNA molecules and precisely cut the DNA at these sequences, causing chain breaks; second, the positions of the two single-strand breaks on the DNA molecule are not always directly opposite, which increases the complexity and diversity of the enzyme cutting reaction; finally, the DNA fragments produced by the cutting of Type II restriction endonucleases often have complementary single-strand extensions at their ends, which facilitates subsequent DNA connection and cloning operations.

The unique properties of plasmid DNA molecules, such as their flexibility and stability in gene manipulation, lay a solid foundation for the implementation of DNA computing. The plasmid DNA computing model cleverly utilizes the gene sites on plasmid DNA molecules, and through the precise action of restriction endonucleases and ligases, enables the plasmid to switch between two different states, which can be symbolically represented by 0 and 1, thus simulating the function of a k bit data register in traditional computers. This innovative concept forms the core principle of plasmid DNA computing. Head and others first put this theory into practice in 2000,

constructing a computing model based on plasmid DNA molecules and successfully applying it to solve the problem of the maximum independent set of vertices in a graph. This pioneering work not only verified the feasibility of plasmid DNA computing but also opened up new directions for subsequent research [8]. Since then, research in this field has continued to deepen, and more research results on plasmid DNA computing models and their applications have emerged [9, 10].

3.1.4 Characteristics of DNA Molecules

This section will introduce some properties of DNA molecules related to DNA computing to the reader. These properties are fundamental in DNA computing research and should be mastered by scholars in the field of DNA computing. The content of this section is introduced for scholars who are not biology majors, but this content is not sufficient. When conducting in-depth research on DNA computing, it is necessary to further master the characteristics and latest results of DNA molecules.

3.1.4.1 Denaturation and Renaturation of DNA Molecules

The denaturation of nucleic acids is a process involving the breaking of hydrogen bonds between base pairs in the double helix region, which can be triggered by physical or chemical factors, leading to the transformation of nucleic acids from a double-stranded structure to a single-stranded form. The biological activity of denatured nucleic acids may be partially or completely lost. It is important to emphasize that nucleic acid denaturation only involves the breaking of hydrogen bonds between bases, while the phosphodiester bonds that maintain the primary structure of nucleic acids remain intact, so the primary structure of nucleic acids remains stable during denaturation. Specifically, the hydrogen bonds in the double-stranded DNA molecule (especially its iconic double helix structure) will break when exposed to conditions such as elevated temperature, extreme changes in medium pH (less than 4 or greater than 10), and the action of specific denaturants (such as organic solvents methanol, ethanol, and chemical reagents such as urea, formamide). When all hydrogen bonds are destroyed, the two polynucleotide chains of the double-stranded DNA molecule will completely separate, a process known as the denaturation or unzipping of the DNA molecule.

The denaturation process can usually be divided into two major categories: one is triggered by a rise in temperature, known as thermal denaturation; the other is caused by changes in the pH of the solution, known as acid-base denaturation. It is worth noting that this biochemical transformation of denaturation actually occurs within a relatively narrow temperature range and is accompanied by significant changes in physical properties, the most critical of which is the change in absorbance characteristics. Specifically, double-stranded DNA in its natural state exhibits lower

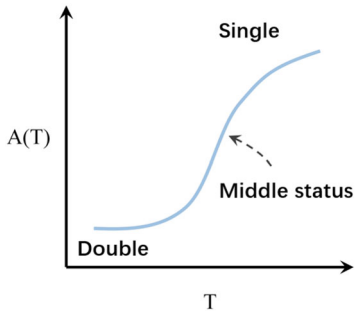


Fig. 3.12 Schematic diagram of the relationship between absorbance and temperature

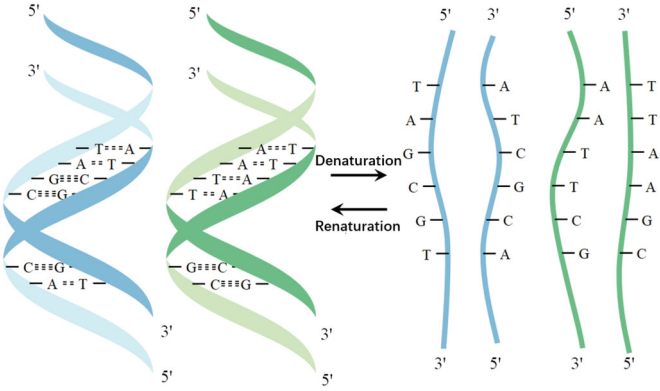


Fig. 3.13 Denaturation and renaturation of DNA molecules

absorbance values compared to an equal amount of single-stranded DNA bases. Therefore, we can indirectly reflect the denaturation process of DNA from double-stranded to single-stranded by monitoring the increase in absorbance values, which is intuitively demonstrated in Fig. 3.12.

Renaturation is the reverse process of denaturation, that is, for two completely complementary single-stranded DNA molecules after denaturation, the process of returning to a double-stranded, or even natural double helix structure under appropriate conditions. Thermally denatured DNA molecules can generally be renatured after cooling, so this process is sometimes also called annealing. The schematic diagram of the denaturation and renaturation process of DNA molecules is shown in Fig. 3.13. The renaturation temperature is generally 25 °C lower than the melting temperature of the DNA molecule T_m value (this concept is introduced in the next section).

After renaturation, the DNA can recover some or all of its physical and chemical properties and biological activity. During the renaturation process, a significant phenomenon is the decrease in ultraviolet absorption value, which is called the hypochromic effect. The efficiency of renaturation is affected by many factors, mainly including the following aspects: First, the cooling rate during renaturation is

crucial and must be kept slow. This is because rapid cooling will hinder the effective collision and binding between DNA molecules, preventing the renaturation process from proceeding fully. Therefore, the process of slowly cooling DNA from high temperature to an appropriate temperature is called annealing, which is beneficial to DNA renaturation. Conversely, if DNA is rapidly cooled from high temperature to low temperature (such as below 4 °C), it is called quenching, under which conditions DNA is difficult to renature. Second, the concentration of DNA is also an important factor affecting renaturation. The higher the concentration, the more opportunities for complementary DNA fragments to collide in space, thereby increasing their chances of binding into double strands, which is beneficial to the renaturation process. Finally, the length of the DNA fragment also affects the difficulty of renaturation. Longer DNA fragments, due to the complexity of their molecular structure, reduce the chances of complementary bases meeting and binding, making the renaturation process more difficult.

3.1.4.2 Melting Temperature

The melting temperature T_m, is defined as the temperature at which 50% of the base pairs in a double-stranded DNA molecule become single-stranded during the denaturation process. It is another important parameter for evaluating the thermodynamic stability of a DNA molecule. The T_m value of a DNA molecule is not only related to its concentration and the pH of the solution, but also to its molecular size and the GC content of the bases it contains, and to the arrangement of the base sequence. Because the melting temperature T_m occupies a very important position in DNA calculations, we introduce it here as a separate section.

One of the main biochemical operations in DNA calculations is the unzipping problem of double-stranded DNA molecules. Since the DNA molecules generally involved in biochemical reactions are massive, and it is required to unzip all the required DNA molecules in a very short period of time, it is required that the DNA molecules as data have as similar or close melting temperatures as possible T_m. To achieve this goal, first, we must require all double-stranded DNA molecules to have the same number of hydrogen bonds when designing DNA sequences, because the denaturation of double-stranded DNA molecules is actually the opening of hydrogen bonds in the double strands; second, considering the impact of base stacking forces, we should also consider the order of DNA sequences when designing DNA sequences. Therefore, how to design DNA sequences to control the melting temperature T_m to keep the melting temperature of all DNA molecules used as data as close as possible is a very important issue. Since Watson and Crick discovered the double helix structure of DNA in 1953 based on Franklin's crystal photos, research on the melting temperature T_m has been ongoing, and current high-throughput sequencing technology also requires large-scale estimates of DNA molecules to ensure sequencing stability and accuracy.

- Empirical formula of short DNA fragments: The calculation formula for the T_m value of oligonucleotide fragments generally less than 20 bp is

$$T_m = 4(G + C) + 2(A + T)$$

  where $G + C$ and $A + T$ are the corresponding base numbers of the DNA molecule.
- Empirical formula based on GC content: In 1962, Marmur and Doty gave the following approximate empirical formula for calculating T_m value [11]: $T_m = 69.3 + 0.41\%(G + C)$, where $\%(G + C)$ is the percentage content of GC bases in the DNA molecule.
- Empirical formula based on DNA concentration: In 1987, Frank-Kamenetskiĭ and others gave the following approximate empirical formula for calculating T_m value [12]

$$T_m = 100.3 + 14.7log_{10}C_0$$

  Where C_0 is the molar concentration of the DNA molecule.
- Calculation formula based on thermodynamic methods: In 1998, SantaLucia summarized the thermodynamic calculation formula for the melting temperature T_m value [13]: $T_m = \triangle H^\circ(\triangle S^\circ + RlnC_t)$, where $\triangle H^\circ$ and $\triangle S^\circ$ are the enthalpy change and entropy change of the hybridization reaction, respectively, R is the gas constant (1.987 cal/Kmol, C_t) is the molar concentration of the DNA molecule. Based on the thermodynamic parameters of neighboring base pairs, this formula can quickly calculate the melting temperature of DNA molecules. In recent years, this thermodynamic formula has been further improved and widely used in large-scale sequencing technology [14].

3.1.4.3 Forces within DNA Molecules

As we have explained before, the key to the formation of stable double-stranded DNA molecules lies in the hydrogen bond forces between bases, especially the hydrogen bonds formed between A and T and between G and C. It is worth noting that because there are three hydrogen bonds between G and C, their mutual attraction is stronger than the two hydrogen bonds between A and T. Therefore, when the length of double-stranded DNA molecules is the same, the GC content directly determines the stability of the double strands: the higher the GC content, the more stable the double strands; otherwise, the stability is worse. In addition, there is another important force within the DNA molecule—the base stacking force, which also plays an important role in the structural stability of DNA. However, the existence of these two main forces also brings some challenges and problems to DNA computation. First, single-stranded DNA molecules can easily form hairpin-like conformations under specific temperature and environmental conditions, which increases the difficulty of specific hybridization. Second, when

designing DNA molecule codes for information processing, we must consider the complex constraints brought by these two forces, which involves at least two aspects of problems, increasing the complexity and challenge of code design.

3.1.4.4 Replication of DNA Molecules

DNA molecules have a powerful ability to replicate, a process that is completed under the catalysis of DNA polymerase, ensuring that a DNA molecule can be accurately replicated into two structurally identical offspring DNA molecules. During the replication process, the original DNA double strand first separates into two single strands, which then each serve as a template, following the strict base pairing principle (A pairs with T, G pairs with C), attracting and connecting the corresponding free nucleotides, thus forming a new chain complementary to the template chain. This replication mechanism ensures that both strands of each DNA molecule can serve as templates for generating new complementary chains. Ultimately, the result of replication is the production of two offspring DNA molecules that are identical to the original DNA molecule in terms of genetic information, as shown in Fig. 3.14.

3.1.5 DNA Biochemical Reactions

The means of information processing in DNA computation is the so-called "specific hybridization" between DNA molecules. DNA computation is actually a series of work carried out for the specific hybridization between DNA molecules, such as coding design problems, solution space design problems, solution detection problems, etc. Because of the key and important role of DNA molecules in DNA

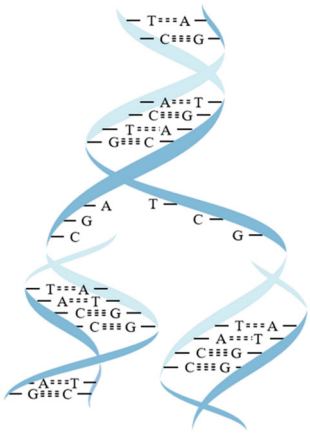


Fig. 3.14 Schematic diagram of DNA replication

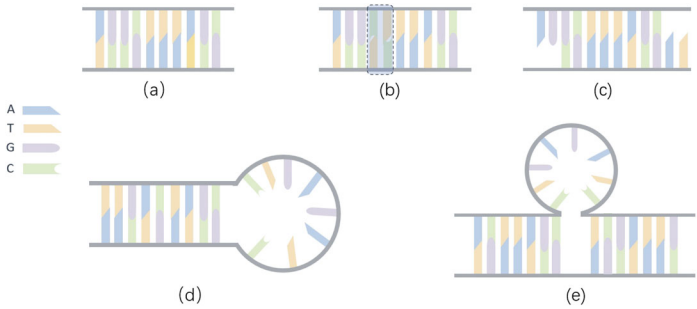


Fig. 3.15 Types of DNA molecule hybridization

computation, we will specifically introduce the related concepts and basic properties of DNA molecule hybridization in this section.

3.1.5.1 Complete Hybridization

Complete hybridization refers to the process where, following the Watson-Crick base pairing principle, two DNA sequences under specific conditions can form stable hydrogen bond connections between their completely complementary base pairs, thus achieving precise molecular matching, as shown in Fig. 3.15a. The core work of the DNA computation field is focused on how to achieve this kind of specific complete hybridization, and on this basis, a series of in-depth research and application explorations are carried out.

3.1.5.2 False Positive Hybridization

False positive hybridization refers to the phenomenon where, under suitable conditions, non-specific binding can occur between DNA molecules that are not completely complementary, forming double-stranded molecules, as shown in Fig. 3.15b and e. The root of this phenomenon lies in the "similarity" between the two DNA molecule sequences involved in the hybridization, leading to unexpected pairing. In DNA computation, false positive hybridization is usually a disadvantageous situation that needs to be avoided as much as possible, because it may interfere with the accuracy of the experimental results. However, this phenomenon is not uncommon in the field of DNA computation and even in the broader field of molecular biology. To effectively overcome the phenomenon of false positive hybridization, the primary strategy is to take preventive measures at the stage of DNA sequence coding design, by optimizing sequence design to reduce the possibility of non-specific binding. Secondly, adjusting experimental conditions is also an important means, such as changing the hybridization temperature, pH value or using specific

hybridization buffers, etc., to improve the specificity of the hybridization reaction. By comprehensively applying these strategies, the occurrence of false positive hybridization can be significantly reduced, improving the accuracy and reliability of DNA computation.

3.1.5.3 False Negative Hybridization

False negative hybridization refers to the phenomenon where completely complementary DNA molecules do not fully hybridize during the reaction process for various reasons, as shown in Fig. 3.15c. We also call this specific phenomenon displacement hybridization. The main reason for the occurrence of false negative phenomena is due to the reaction conditions and mistakes in the biochemical operation itself. Therefore, we need to operate carefully when conducting biochemical experiments.

3.1.5.4 Hairpin Hybridization

Hairpin hybridization refers to a special phenomenon where a single DNA strand, under specific conditions of its own base sequence, undergoes internal folding due to hydrogen bond attraction, forming a local double-stranded structure, as shown in Fig. 3.15d. This self-hybridization phenomenon is usually undesirable in most cases, but through careful biochemical operation, we can effectively control and overcome it. However, in some cases, hairpin hybridization can be cleverly used in DNA computation to achieve specific information processing functions. For example, in the DNA computation model for solving satisfiability problems described in reference [15], hairpin DNA is used to mark non-solutions, and by introducing enzyme cutting sites on the hairpin structure, non-solutions are removed from the computation system. In addition, regarding the formation of hybrid double strands, it is not only limited to DNA-DNA, but can also occur between RNA-DNA strands, and even between PNA (peptide nucleic acid) and DNA. Given that current DNA computation, RNA computation or PNA computation technologies have not widely utilized these types of hybridization, this article will not discuss them in depth.

3.2 RNA Molecules

RNA molecules are similar in composition to DNA molecules, also possessing four types of base information molecules (see Fig. 3.16). The nucleotide components are polymerized by covalent bonds to form high molecular weight chains. There are many types of RNA in living organisms, generally existing in single-strand form, except for short double-strand RNA, virus double-strand RNA, and cyclic under certain physiological conditions. Currently known molecular forms

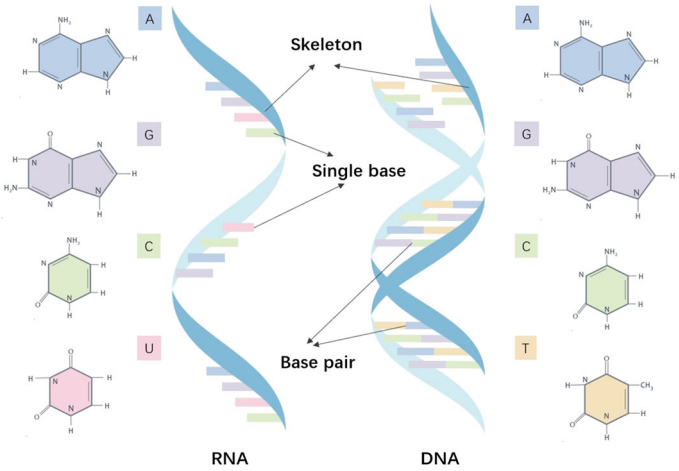
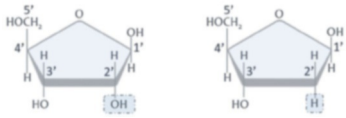


Fig. 3.16 Molecular composition of RNA and DNA

Fig. 3.17 Ribose (left) and Deoxyribose (right)



include messenger RNA (mRNA), transfer RNA (tRNA), ribosomal RNA (rRNA), microRNA (miRNA), small interfering RNA (siRNA), short hairpin RNA (shRNA), long non-coding RNA (lncRNA), etc., participating in gene transcription, protein expression, developmental regulation, immune regulation, and other life activities. Based on the characteristics of RNA information flow, various RNA calculation and detection technologies are also booming, including genome editing technology based on guide RNA (gRNA) for the development of new devices [16]. This section mainly introduces the differences in the composition of RNA and DNA molecules, the basic structure of RNA, the types of RNA molecules, and the RNA calculation model. For details, see Chap. 11.

3.2.1 Nucleotides

The nucleotide of RNA is formed by the condensation of a molecule of phosphoric acid, a molecule of pentose, and a molecule of organic base, in which the pentose is ribose, while the pentose in DNA is 2-deoxyribose (see Fig. 3.17).

The organic bases in RNA are also divided into two types: purines and pyrimidines. The purines are the same as DNA, divided into adenine and guanine; pyrimidines are divided into cytosine and uracil (uracil, U), without the thymine

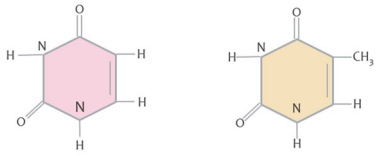


Fig. 3.18 Uracil (left) and Thymine (right)

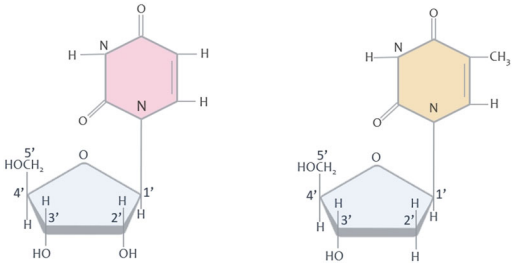


Fig. 3.19 Uracil nucleoside (left) and thymine deoxynucleoside (right)

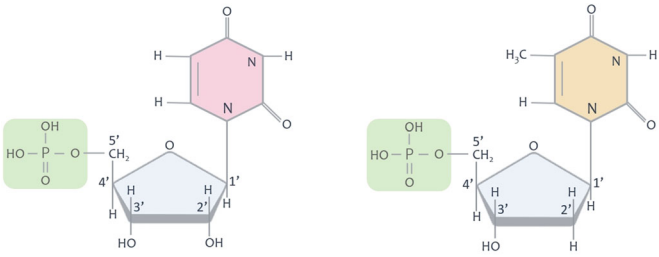


Fig. 3.20 5'-Uracil nucleotide (left) and 5'-thymine deoxynucleotide (right)

in DNA (Fig. 3.18). The character set of RNA as a coding information carrier is AUGC, which is slightly different from the DNA character set ATGC.

Similar to DNA, the 1st C atom on the pentose molecule in RNA is connected to the 9th N atom on the purine or the 1st N atom on the pyrimidine by a β type C-N glycosidic bond to form a nucleoside. There are also four types of ribonucleosides in RNA: adenine nucleoside (adenosine), guanine nucleoside (guanosine), uracil nucleoside (uridine), cytosine nucleoside (cytidine). Figure 3.19 shows the structural differences between uracil nucleoside and thymine deoxynucleoside.

Like DNA, the site where the phosphate binds to the ribose in RNA is usually the 3' or 5' carbon atom of the ribose. Corresponding to the four nucleosides, there are also four nucleotides in the RNA molecule. Figure 3.20 shows the structural differences between 5'-uracil nucleotide (5'-UMP) and 5'-thymine deoxynucleotide (5'-dTMP). The pentose ring in the RNA nucleoside molecule is phosphorylated to form a nucleotide.

3.2.2 RNA Molecular Structure

Each nucleotide in RNA contains a ribose (carbon numbered from 1' to 5'), with adenine (A), cytosine (C), guanine (G), or uracil (U) attached to the 1' position of the ribose. The ribose and phosphate polymerize into the backbone of the RNA chain through phosphodiester bonds. An important structural difference between RNA and DNA is the presence of a hydroxyl group at the 2' position of the ribose. RNA single strands can form local double strands, and the 2' hydroxyl group makes the RNA double-strand structure different from the DNA structure [17]. RNA molecules are flexible in conformation, some regions do not participate in the formation of the double helix structure, and are easily affected by the cutting effect of nucleases [18].

RNA only has four bases (adenine, cytosine, guanine, and uracil), but during the process of RNA transcription and selective splicing, bases can be covalently modified in various ways to form special nucleotides that participate in RNA synthesis and play special physiological roles. The covalent bond between uracil and ribose in pseudouridine (Ψ) changes from a C-N bond to a C-C bond, and methyluridine is prominent in the T Ψ C loop of tRNA [19]. Hypoxanthine is also a modified base, specifically a deaminated adenine base, its nucleoside is called inosine, which plays a key role in the wobble hypothesis of the genetic code [20]. There are also more than 100 other naturally occurring modified nucleosides in nature, among which the structural diversity of modifications in tRNA and rRNA is the most frequent, but their specific physiological functions are not yet fully understood.

Similar to DNA, single-stranded RNA molecules also have primary, secondary, and tertiary structures. The primary structure is the condensation of nucleotide components, obtained by the sequential arrangement of four nucleotides (Fig. 3.21a). The secondary structure elements are the result of intramolecular hydrogen bonding, such as hairpins, stem loops, and free single-strand regions (Fig. 3.21b). The tertiary structure is further twisted and wound on the basis of the secondary structure, forming a perfect functional domain. Like proteins, RNA in living organisms often requires a specific tertiary structure to perform molecular functions.

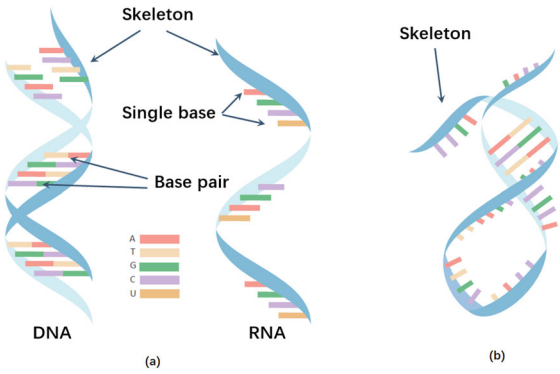


Fig. 3.21 RNA sequence and structure

3.2.3 Types of RNA Molecules

There are many types of RNA in living organisms. Based on the length of the RNA chain, RNA can be simply divided into small RNA and long RNA [21]. Small RNA is less than 200nt in length, while long RNA is longer than 200nt. Currently, long RNA mainly includes long non-coding RNA (lncRNA) and mRNA, while small RNA mainly includes 5.8S ribosomal RNA (rRNA), 5S rRNA, transfer RNA (tRNA), microRNA (miRNA), small interfering RNA (siRNA), small nucleolar RNAs (snoRNAs), Piwi-interacting RNA (piRNA), tRNA-derived small RNA (tsRNA), and small rDNA-derived RNA (srRNA).

Messenger RNA (mRNA) is a type of RNA that carries genetic information. The information flow from the DNA coding region enters the ribosome, and the messenger RNA can be considered a genetic copy of the DNA. According to the biological rule that each three nucleotides (codon) corresponds to one amino acid, the coding sequence of mRNA determines the amino acid sequence of the corresponding protein. In eukaryotic cells, once precursor mRNA (pre-mRNA) is transcribed from DNA, it is processed into mature mRNA. The mRNA then exits the nucleus to the cytoplasm, where it binds with the ribosome and is translated into the corresponding protein form with the help of tRNA. In prokaryotic cells without a nucleus and cytoplasmic compartments, mRNA can bind with the ribosome during transcription from DNA, and then degrade into its constituent nucleotides with the help of intracellular ribonuclease.

Unlike mRNA, the most prominent examples of non-coding RNA are transfer RNA (tRNA) and ribosomal RNA (rRNA), both of which are involved in the translation process. tRNA (Transfer RNA) is a small RNA chain composed of about 80 nucleotides, which transfers specific amino acids to the growing polypeptide chain at the protein synthesis ribosome site during the translation process. It has an amino acid attachment site and an anticodon region for codon recognition, which binds to a specific sequence on the messenger RNA chain through hydrogen bonds. Ribosomal RNA (rRNA) is a component of the ribosome. Eukaryotic ribosomes contain four different rRNA molecules: 18S, 5.8S, 28S, and 5S rRNA. Ribosomal RNA and proteins combine to build the ribosome complex, which further binds with mRNA to carry out intracellular protein synthesis. Compared with other RNAs, rRNA is the most abundant in eukaryotic cells, corresponding to the basic physiological function of protein translation.

In addition to protein regulatory factors such as inhibitors and activators, current research shows that RNA also regulates genes. The RNA regulation mechanisms in eukaryotes are diverse, such as RNAi inhibiting genes post-transcriptionally, long non-coding RNA shutting down chromatin blocks epigenetically, and enhanced RNA-induced gene expression increasing. Bacteria and archaea have also been proven to use regulatory RNA systems, such as bacterial small RNA and CRISPR [22]. microRNAs are RNA molecules that regulate mRNA translation through the principle of complementary pairing. With the diversification of RNA regulatory

mechanisms, computational theorists can use the flow of genetic information to build mathematical models of specific problems and construct nanoscale computers.

3.3 Protein Molecules

Proteins in nature are composed of 20 common amino acids linked sequentially by covalent bonds, and further form secondary, tertiary, and quaternary structures through non-covalent interactions such as hydrogen bonds, electrostatic interactions, hydrophobic interactions, and van der Waals forces. These structures play important roles in organisms and are crucial for signal perception. A single protein sequence can form multiple conformations, and changes in protein conformation can represent changes in signals, forming the basis for computational models. Compared to nucleic acid molecules, protein molecules have diverse structures and a wide range of functions. Detailed information on protein computation can be found in Chap. 12. This section mainly introduces protein structure, classification, and protein computation output detection technology.

3.3.1 Protein Structure

Amino acids are the basic units that make up proteins. They are organic acids with amino groups. Their general structure is shown in Fig. 3.22. They consist of an amino group, a carboxyl group, a hydrogen atom, and an R group (R group is a variable group). Based on different R groups, there are 20 basic amino acids that make up various proteins in organisms [21]. Protein molecules are covalent polypeptide chains formed by the dehydration condensation of amino acids. Proteins have primary, secondary, tertiary, and quaternary structures. The structure of a protein molecule determines its function. This book uses hemoglobin as an example to introduce the primary, secondary, tertiary, and quaternary structures of proteins.

The primary structure of a protein is the order of amino acid residues in the protein polypeptide chain. Each protein has a unique and precise amino acid sequence, which is determined by the order of genetic codes on the gene and is the most basic structure of a protein. Hemoglobin is a metalloprotein that binds

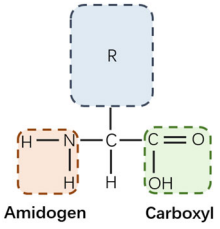


Fig. 3.22 General structure of amino acids

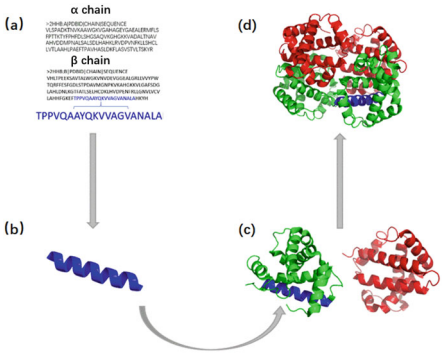


Fig. 3.23 Hemoglobin structure

to oxygen with iron ions. It consists of two non-covalently bound α and two β subunits, which are generally found in the red blood cells of vertebrates. Its main function is to bind and transport oxygen in the blood. Figure 3.23a shows the amino acid sequences of the α and β subunits of human hemoglobin, which consist of 141 and 146 amino acid residues, respectively. The most authoritative protein primary structure or protein sequence database internationally is UniProt (https://www.uniprot.org/), which has collected more than 560,000 manually annotated protein sequences. Protein molecules are not linearly extended, but fold and curl into relatively stable spatial structures to perform their biological functions. The spatial structure of a protein refers to its secondary, tertiary, and quaternary structures.

The secondary structure of a protein refers to a specific local spatial regular conformation formed by a certain segment of the peptide chain backbone atoms through hydrogen bonds along a certain axis, without involving the side chains of amino acid residues. The secondary structure of proteins mainly includes α-helices, β-sheets, and β-turns. The polypeptide chain of the α-helix spirals regularly around the central axis, with every 3.6 amino acid residues spiraling up one turn, moving up 0.54 nm. After the adjacent amino acid residues form a complete α-helix, the subsequent residues will form this helical structure more easily and quickly. The α-helix is the most common, typical, and abundant secondary structure in proteins. About 35% of the amino acids in natural proteins are located in the α-helix structure. For example, 68% and 70% of the amino acids in the α and β subunits of hemoglobin, respectively, are located in the α-helix structure. Figure 3.23b is a segment of the α-helix of the β subunit of hemoglobin. β-sheets are also repetitive structures, which can be roughly divided into parallel and antiparallel types. They are maintained by hydrogen bonds between peptide chains or segments, and the peptide bond plane folds into a sawtooth shape, with regular hydrogen bonds formed between the N-H and C=O of the adjacent peptide chain backbone. The 180° turn structure that appears in the peptide chain is called a β-turn, which is composed of four consecutive amino acid residues. The second amino acid residue is often proline, and the carbonyl of the first amino acid residue forms a hydrogen bond with the amine of the fourth amino acid residue to maintain its stability. Irregular curls, in addition to α-helices, β-sheets, and β-turns, are often irregular secondary structures

that have important biological functions, but relatively irregularly arranged rings or curl structures.

The tertiary structure of a protein refers to the specific spatial structure formed by a polypeptide chain further winding and folding on the basis of various secondary structures, maintained by secondary bonds. The stability of the tertiary structure of proteins mainly depends on secondary bonds such as hydrogen bonds, hydrophobic bonds, salt bonds, and van der Waals forces. These secondary bonds can exist between the R groups of amino acid residues far apart in the primary structure of proteins. In addition, disulfide bonds in most proteins also play a very important role in the stability and formation of the tertiary structure. Figure 3.23c shows the tertiary structure of the α and β subunits of hemoglobin.

Many biologically active proteins are composed of two or more polypeptide chains with tertiary structures. Each polypeptide chain is called a subunit, and the spatial relationship between subunits is maintained by non-covalent bonds, forming the quaternary structure of the protein. The binding force between subunits is mainly hydrophobic bonds, and hydrogen bonds and ionic bonds also participate in maintaining the quaternary structure. For example, hemoglobin (Fig. 3.23d) is composed of four polypeptide chains with tertiary structures, two of which are α-chains and the other two are β-chains, and its quaternary structure is approximately ellipsoidal.

3.3.2 Protein Classification

Proteins can be divided into globular proteins and fibrous proteins according to their molecular shape. The former is approximately spherical in shape, soluble in water and active, such as enzymes, transport proteins, protein hormones, antibodies, etc. The latter is generally slender in shape, has a large molecular weight, and is mostly structural proteins, such as collagen. Fibrous proteins can be divided into soluble fibrous proteins and insoluble fibrous proteins. The former includes fibrinogen in the blood, myosin in the muscles, etc., and the latter includes keratin and other structural proteins.

According to the molecular composition, proteins can be divided into simple proteins (pure proteins) and conjugated proteins. The former is composed entirely of amino acids and does not contain non-protein components, such as serum albumin. Simple proteins can be further divided into 7 types based on solubility: albumin, globulin, histone, protamine, gluten, alcohol-soluble protein, and hard protein. The globulin here is different from the globular protein mentioned earlier. It refers to a pure protein that is insoluble or slightly soluble in water and soluble in dilute salt solution. Immunoglobulin is one type of globulin. Conjugated proteins, in addition to the peptide chains composed of amino acids, also contain non-protein components, collectively referred to as prosthetic groups. According to the different prosthetic groups, conjugated proteins can be divided into nucleoproteins,

lipoproteins, glycoproteins, phosphoproteins, heme proteins, flavin proteins, and metalloproteins.

Proteins can also be classified according to their different functions in organisms, and can be divided into enzymes, structural proteins, transport proteins, immune proteins, recognition proteins, and other functional proteins. Enzymes are the most common type of protein, they catalyze biochemical reactions, can accelerate the speed of various chemical reactions, and are especially important for the metabolism of organisms. Structural proteins mainly undertake the construction of internal and external structures of cells. Transport proteins act as transport proteins, responsible for transporting various substances inside and outside cells. Immune proteins can act as secretions of immune cells, helping the body fight against pathogens. Recognition proteins are special structures on the surface of cells, used for cell recognition. Other functional proteins include some proteins that have not yet been clearly defined, and they may play a variety of roles in organisms.

The above is a classification based on the role of proteins in organisms, but in fact, each protein may be a combination of several of the above functions, because the structure of proteins themselves is complex and diverse, and they can perform multiple tasks under different physiological conditions [21].

3.3.3 Detection Technology

The output of protein calculation usually manifests as specific biochemical effects or biological effects, and needs to be detected with the help of existing biological analysis techniques. Optical technology plays a crucial role in the analysis of output signals of protein calculation. The most commonly used optical technologies for analyzing output signals of protein calculation include light absorption, fluorescence spectroscopy, and surface plasmon resonance. The following introduces them separately.

In the method of light absorption, the intensity of light will weaken after passing through a solution or substance. The logarithm of the ratio of the incident light intensity before and after passing through is called absorbance. By measuring the change in the absorbance of the sample to light of a specific wavelength, the composition and concentration of chemical substances in the sample can be analyzed. In the research of protein calculation, the output signal of protein calculation can be detected by measuring the change in the absorption intensity of the produced or consumed substance to light of a specific wavelength.

Fluorescence spectroscopy is a phenomenon of photoluminescence. When a molecule absorbs light of a specific wavelength, it enters an excited state. In this state, the electronic structure within the molecule changes, causing some electrons to transition to higher energy levels. When these electrons transition back to the ground state, they release additional energy, resulting in the emission of visible light. By measuring the fluorescence spectrum emitted by a sample after being irradiated with excitation light, the chemical substances in the sample can be

detected. In protein computation research, the output signals of protein computation can be analyzed by measuring the fluorescence spectrum of fluorescent substances produced or consumed in the sample.

Surface plasmon resonance is a characterization technique based on optical principles, often used to study the interactions of biomolecules and conformational changes of membrane proteins. The sample to be tested is connected to the metal surface, and polarized light or total reflection light is coupled to the metal surface. When the angle or wavelength of the incident light matches the plasmon resonance frequency on the metal surface, a resonance phenomenon occurs, causing changes in light absorption or reflection. By measuring the plasmon resonance phenomenon between the sample and the metal surface, the chemical substances in the sample can be analyzed. In protein computation research, the sample can be combined with the metal surface, and the output signals of protein computation can be analyzed by measuring the plasmon resonance phenomenon between the sample and the metal surface.

All of the above methods are based on optical principles, measuring the absorption, emission, scattering, etc., of light by the sample to achieve qualitative and quantitative analysis of the chemical substances in the sample [23]. In protein computation research, these methods can help researchers monitor and analyze the output signals of reactions in real time, thereby evaluating their computational effects and performance.

References

1. Fang, G., Zhang, S., Dong, Y., et al.: A novel DNA computing model based on RecA-mediated triple-stranded DNA structure. Progress in Natural Science. **17**, 708–711 (2007)
2. Xu, J., Zhang, S.M., Fan, Y.K., Guo, Y.M.: Principles, progress and difficulties of DNA computer (III). Journal of Computer Science. **30**, 869–880 (2007)
3. Benenson, Y., Gil, B., Ben-Dor, U., et al.: An autonomous molecular computer for logical control of gene expression. Nature. **429**, 423–429 (2004)
4. Roweis, S., Winfree, E., Burgoyne, R., et al.: A Sticker-Based Model for DNA Computation. Journal of Computational Biology. **4**, 615–629 (1998)
5. Xu, J., Dong, Y.F., Wei, X.P.: Sticker DNA computer model (I): Theory. Chinese Science Bulletin. **49**, 1–8 (2004)
6. Xu, J., Li, S.P., Dong, Y.F., et al.: Sticker DNA computer model (II): Applications. Chinese Science Bulletin. **49**, 1–9 (2004)
7. Wu, N.H.: Principles of Genetic Engineering (Second Edition). Science Press, Beijing (2002)
8. Head, T., Rozenberg, G., Bladergroen, R.B., et al.: Computing with DNA by operating on plasmids. BioSystems. **57**, 87–93 (2000)
9. Zhang, L.Z, Liu, G.W., Xu, J.: Research on DNA computing model based on plasmid. Computer Engineering and Applications. **4**, 51–52 (2004)
10. Ouyang, Q., Kaplan, P., Liu, S.: DNA Solution of the Maximal Clique Problem. Science. **278**, 446–449 (1997)
11. Marmur, J., Doty, P.: Determination of the base composition of deoxyribonucleic acid from its thermal denaturation temperature. J. Mol. Biol. **5**, 109–118 (1962)
12. Frank-Kamenetskiĭ, M., Anshelevich, V., Lukashin, A.: Polyelectrolyte model of DNA. Sov. Phys. Usp. **30**, 317–330 (1987)

13. SantaLucia Jr, J.: A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proc. Natl. Acad. Sci. USA. **95**, 1460–1465 (1998)
14. Chen Y, Huang X.: DNA Sequencing By Denaturation: Principle and Thermodynamic Simulations. Anal Biochem. **384**, 170–179 (2009)
15. Sakamoto, K., Gouzu, H., Komiya, K., et al.: Molecular computation by DNA hairpin formation. Science. **288**, 1223–1226 (2000)
16. Shipman, S., Nivala, J., Jeffrey, D., et al.: CRISPR—Cas encoding of a digital movie into the genomes of a population of living bacteria. Nature. **547**, 345–349 (2017)
17. Salazar, M., Fedoroff, O.Y., Miller, J.M., et al.: The DNA strand in DNA.RNA hybrid duplexes is neither B-form nor A-form in solution. Biochemistry. **32**, 4207–4215 (1993)
18. Mikkola, S., Stenman, E., Nurmi, K., et al.: The mechanism of the metal ion promoted cleavage of RNA phosphodiester bonds involves a general acid catalysis by the metal aquo ion on the departure of the leaving group. Journal of the Chemical Society, Perkin Transactions. **2**, 1619–1626 (1999)
19. Yu, Q., Morrow, C.: Identification of critical elements in the tRNA acceptor stem and T(Psi)C loop necessary for human immunodeficiency virus type 1 infectivity. Journal of Virology. **75**, 4902–4906 (2001)
20. Elliott, M., Trewyn, R.: Inosine biosynthesis in transfer RNA by an enzymatic insertion of hypoxanthine. The Journal of Biological Chemistry. **259**, 2407–2410 (1984)
21. Shen, T., Wang, J.Y.: Biochemistry (Third Edition). Higher Education Press, Beijing (2002)
22. Gottesman, S.: Micros for microbes: non-coding regulatory RNAs in bacteria. Trends in Genetics. **21**, 399–404 (2005)
23. Sambrook, J., Russell, D.: Molecular Cloning: A Laboratory Manual (Third Edition). Cold Spring Harbor Laboratory, New York (2000)

Chapter 4
Biological Computing Operators: Enzymes and Biochemical Operations

The previous chapter introduced the data required for biological computing: DNA, RNA, and proteins. They form the basis of biological computing. This chapter presents another important foundation of biological computing—basic operators: biological enzymes and biochemical operations.

4.1 Commonly Used Enzymes in Biological Computing

Biological computing is developing very rapidly, it has its own unique information processing system, and the operations that execute these information processing cannot be separated from the tool enzymes (essentially proteins) that mediate biochemical reactions. These tool enzymes are introduced as follows, and the specific use of enzymes will also be introduced in Sect. 4.2.

4.1.1 Restriction Endonucleases

Restriction endonucleases (also known as restriction enzymes) are important cutting tools commonly used in biochemical reaction operations in biological computing. These enzymes were first discovered in certain strains of E. coli, which can "restrict" phage infection, hence the name. These enzymes can specifically recognize and attach to specific nucleotide sequences, and cut the phosphodiester bond between two deoxyribonucleotides at a specific location on each strand. This cutting reaction usually occurs at a specific nucleotide sequence, which is a palindromic sequence. It refers to a sequence where the base order read forward on one strand is exactly the same as the order read backward on the other strand. According to the structure of the restriction enzyme, the requirement of cofactors, the cutting site and the mode

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_4

of action, restriction enzymes can be divided into three types, namely Type I, Type II, and Type III.

1. Type I, which has both modification and restriction cutting functions; it also has the ability to recognize specific base sequences on DNA, usually its cutting site is thousands of bases away from the recognition site, and it cannot accurately locate the cutting site, so it is not commonly used. For example: EcoB, EcoK.
2. Type II, which only has the function of restriction cutting, and the modification function is performed by other enzymes. The recognized position is often a short palindromic sequence; the cut base sequence is usually the recognized sequence. It is a type of restriction enzyme with high practicality in genetic engineering. For example: EcoRI, HindIII.
3. Type III, similar to Type I restriction enzymes, has both modification and recognition cutting functions, recognizes short asymmetric sequences, and the cutting site is 24–26 base pairs away from the recognition sequence, and it cannot accurately locate the cutting site, so it is not commonly used. For example: EcoPI, HinfIII.

The most commonly used in biochemical reaction operations in biological computing is Type II restriction endonucleases, which are often used as "cutting" operators. According to the cutting method, restriction enzymes can be divided into staggered cuts and blunt cuts. Staggered cuts are generally cut at different parts of the two strands, with a few nucleotides in between. After cutting, the two ends form a palindromic single-strand end. This end can be connected with the DNA fragment of the target gene with complementary bases, so it is called a sticky end. The other is to cut at the same position of the specific sequence of the two strands to form a blunt end.

The naming of restriction enzymes is determined by the type of bacteria, such as EcoR as an example: E is the capitalized initial of Escherichia, representing the genus of the discovered bacteria, co is the abbreviation of coli, representing the species of the discovered bacteria, R is the capitalized initial of RY13 representing its strain, I is the Roman numeral "one", indicating the first restriction enzyme discovered in this type of bacteria, representing the order of discovery. Other common restriction enzymes and their recognition sequences [1] can be found in the appendix of this chapter.

4.1.2 Polymerase

Polymerase (DNA Polymerase) is an enzyme involved in DNA replication and synthesis. In 1957, American scientist Arthur Kornberg first discovered DNA polymerase in E. coli, which is known as DNA polymerase I (DNA polymerase I, abbreviated as: Pol I). In 1970, German scientist Rolf Knippers discovered DNA polymerase II (Pol II). Subsequently, DNA polymerase III (Pol III) was discovered. The main DNA polymerase in prokaryotes and responsible for chromosome repli-

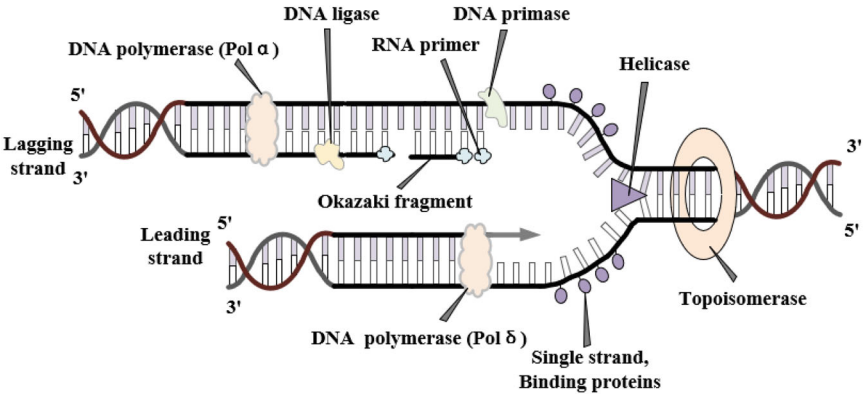


Fig. 4.1 DNA replication

cation is Pol III. It mainly catalyzes the polymerization of deoxyribonucleotides in the form of a template. The polymerized molecules will form a template chain and further participate in pairing. It plays a key role in biological computing systems and is widely used in the polymerase chain reaction technology PCR (Polymerase Chain Reaction) based on it, which we will discuss in detail in Sect. 4.4 of this chapter.

DNA polymerase uses deoxynucleotide triphosphates (dATP, dCTP, dGTP, or dTTP, collectively referred to as dNTPs) as substrates, along the $3' \rightarrow 5'$ direction of the template, connects the corresponding deoxynucleotide to the $3'$ end of the original DNA chain, extending the new chain in the $5' \rightarrow 3'$ direction. The sequence of the new chain is complementary to the sequence of the original template chain and is consistent with the sequence of the original paired chain. All known DNA polymerases synthesize DNA in the $5' \rightarrow 3'$ direction and cannot "renew" (de novo) synthesize DNA, but can only add deoxynucleotides to the $3'$ end of existing RNA or DNA hydroxyl. Therefore, in addition to needing a template as a sequence guide, DNA polymerase also needs a primer to initiate synthesis. The enzyme that synthesizes the primer is called primase. DNA synthesis mediated by DNA polymerase starts with the pairing of the primer and DNA. The paired primer has a free hydroxyl at the $3'$ end. Subsequently, under the catalysis of DNA polymerase, the paired electrons on the free hydroxyl oxygen attack the phosphorus atom on the triphosphate base and nucleophilically replace it, thereby forming an ester bond between the pentose and phosphate to complete the extension of a base. During the entire process, the energy is provided by the high-energy phosphate bond carried by the triphosphate base. After the formation of the phosphate ester, a pyrophosphate molecule is released, and the pyrophosphate molecule splits again to provide sufficient energy for the DNA polymerization process.

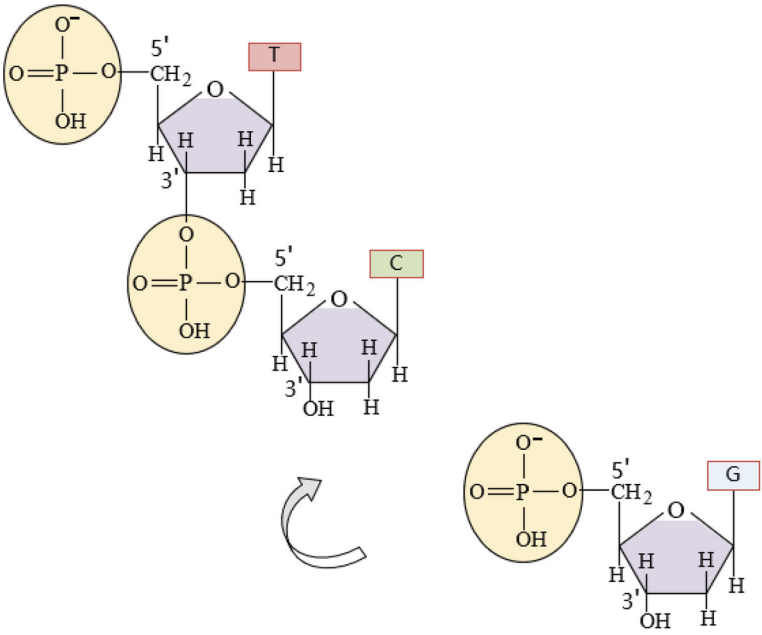


Fig. 4.2 Nucleophilic substitution of DNA chain extension

The main types of DNA polymerase are:

1. In prokaryotes.
   (a) DNA Polymerase I (Pol I): The DNA polymerase I of E. coli K-12 strain is encoded by the polA gene, composed of 928 amino acids, with a molecular weight of 103.1kDa, similar to a spherical structure, with a diameter of about 6.5 nm, and there are about 400 such polymerase molecules in each bacterial cell. It was also the first polymerase attempted for PCR.
   (b) DNA Polymerase II (Pol II): It plays a role in damage repair during DNA stabilization.
   (c) DNA Polymerase III (Pol III): It plays a major role in the DNA replication process of E. coli.
   (d) DNA Polymerase IV (Pol IV): Works with DNA Polymerase II for damage repair during the stable period.
   (e) DNA Polymerase V (Pol V): Participates in SOS repair.
   (f) Family D DNA Polymerase.
2. In eukaryotes.
   a. Pol α: Forms a complex with DNA Primase (Pol α-primase complex), synthesizes about 10nt RNA primers, then extends this RNA primer as a

DNA polymerase; after synthesizing about 20 bases (iDNA), it hands over the subsequent extension process to Pol δ and ϵ.

b. Pol β: Plays a role in DNA repair, low fidelity replication.
c. Pol γ: Replicates mitochondrial DNA.
d. Pol δ: Pol δ and Pol ϵ are the main DNA polymerases of eukaryotic cells. Used for lagging strand synthesis.
e. Pol ϵ: Fills primer gaps, excision repair, recombination, used for leading strand synthesis.
f. Pol ζ: Participates in translesion DNA synthesis (TLS), especially in the extension of primer DNA after bypassing DNA damage [2].

Another very important DNA polymerase is Taq DNA polymerase (Thermus aquaticus). Because it is very important in biological computation, it is the basis of PCR reaction, so it is listed separately and described in detail.

Kary Mullis began in 1983 to try to add DNA polymerase after hybridizing two primers with the target DNA fragment, this method can achieve exponential DNA replication. After each round of replication, the mixture needs to be heated to above 90 °C to melt the newly synthesized DNA; the two DNA strands can only become the template for the next round of replication after separation. Before the discovery of Taq enzyme, the heating process would also inactivate the E. coli DNA polymerase I used at the time. The application of Taq enzyme allows PCR to be carried out at high temperatures (~60 °C), which helps to improve primer specificity and reduce non-specific products. PCR only needs to be carried out on a relatively simple thermal cycler in a closed test tube. Therefore, Taq enzyme is the cornerstone of solving many problems related to DNA analysis in molecular biology, and it is also the cornerstone of the initial biological computation.

Taq polymerase was isolated from thermophilic bacteria Thermus aquaticus by Chien Chia-yun in 1976 [3]. Common abbreviations for Taq polymerase include Taq Pol (or Taq enzyme). Thermus aquaticus lives in hot springs and deep-sea hot springs. The Taq enzyme isolated from it can withstand the high temperatures required for PCR [4]. Therefore, Taq enzyme replaced the original E. coli DNA polymerase used in PCR [5]. The full length of the Taq enzyme gene is 2496 bases, with 832 amino acids, a molecular weight of 94 kDa, the optimal activity temperature is 75–80 °C, the half-life is more than 2 hours at 92.5 °C, 40 minutes at 95 °C, and 9 minutes at 97.5 °C; Taq enzyme can replicate a DNA containing 1000 base pairs in 10 seconds at 72 °C [6].

One of the disadvantages of Taq enzyme is the lack of $3'$ to $5'$ exonuclease proofreading activity, which results in low fidelity of Taq in replication. The original error rate was 1 error per 9000 nucleotides [7].

In order to reduce the chance of errors, scientists have successively discovered other polymerases that can replace Taq enzyme. For example, Pfu is a polymerase with $3'$ to $5'$ exonuclease characteristics, and the error rate is about 1/26000000. But compared to Taq polymerase, Pfu synthesizes DNA more slowly, so there are also mixed use formulas being made.

In addition to fast synthesis speed and high error rate, Taq polymerase will also make the synthesized product "end with an A base", TA cloning is to use this characteristic of Taq polymerase, Taq's PCR product will have an extra A at the $3'$ end, at this time only a complementary T is displayed on the carrier, they can approach each other, and connect by ligase. Through this method, the time for using restriction enzymes to cut can be saved, and the PCR product and the carrier can be quickly glued together directly using the characteristic of complementary ends.

Factors affecting the reaction activity of Taq enzyme:

(1) Temperature: Although Taq DNA polymerase has a strong temperature adaptation range, an environment higher than 60 °C will still denature and inactivate some enzymes. Conversely, if the temperature is lower than normal, enzyme activity is restricted. Moreover, because the primer may bind to homologous sequences in other parts of the genome at low temperatures (especially 25–27 °C), some amplification products are not the target sequence. Appropriately increasing the temperature, mismatched bases will dissociate, and the specificity of the reaction product will increase. The optimal temperature for Taq DNA polymerase is about 70 °C;

(2) Magnesium Ion Concentration: The activity of Taq DNA polymerase is very sensitive to the concentration of Mg^{2+}. Like many other polymerases, Taq DNA polymerase is a Mg^{2+}-dependent enzyme. Using salmon sperm DNA as a template, the total concentration of dNTPs is 0.9 $\sim$0.8 mmol/L, and the PCR system with different concentrations of $MgCl_2$ is used to carry out the reaction for 10 minutes. The results show that under the condition of 2.0 mmol/L $MgCl_2$, the enzyme activity shows an increase. If the concentration of Mg^{2+} is too high, the enzyme activity will be limited, and 10 mmol/L $MgCl_2$ inhibits the enzyme activity by about 50%. Since Mg^{2+} can bind with negative ions or negative ion groups (such as phosphate groups), in PCR, DNA templates, primers, and dNTPs are the main sources of phosphate groups, among which dNTPs occupy a large proportion. Therefore, in the reaction system, the optimal concentration of Mg^{2+} is also affected by the concentration of dNTPs. To obtain the best reaction results, it is necessary to explore the reaction conditions. Whenever a new target fragment and primer are used for the first time, or when a certain parameter (dNTP or primer concentration) changes, the optimal concentration of Mg^{2+} should be titrated. A general principle is that the final concentration of Mg^{2+} in the sample should be at least 0.5 $\sim$1.0 mmol/L higher than the total concentration of dNTPs.

(3) KCl concentration: Generally it is 50 mmol/L, and when it is higher than 75 mmol/L, the PCR reaction is significantly limited. When the KCl concentration reaches 200 mmol/L or more, the PCR reaction is significantly affected, and there is still no nucleotide insertion after the reaction for 10 minutes. The effects of NH_4Cl, NH_4Ac and NaCl, all at a concentration of 50 mmol/L, on the activity of Taq DNA polymerase are moderate inhibition, no effect, and 25 $\sim$30% promotion, respectively.

(4) dNTP concentration: Balanced low concentration dNTPs are more conducive to the exertion of enzyme activity, can reduce mismatches, and obtain a large amount of highly specific DNA reaction products. A 100 μl PCR system with a nucleotide concentration of 40 μmol/L can produce 2.6 μg of DNA product, consuming only half of the provided nucleotides.
(5) Denaturants: 10% ethanol does not inhibit enzyme activity; dimethyl sulfoxide (DMSO), dimethyl formamide (DMF). Formamide has no effect on enzyme activity at low concentrations, but as their concentrations increase, enzyme activity decreases significantly. Ten percent DMSO can halve enzyme activity. However, other researchers have observed that 10% DMSO plays a beneficial role in some reaction systems. This diverse phenomenon is also observed in experiments using urea. 1.0 mol/L urea can enhance enzyme activity, 2.0 mol/L urea can preserve most enzyme activity. However, there are also reports that 0.5 mol/L urea completely inhibits PCR. In summary, the impact parameters of denaturants on Taq DNA polymerase and the PCR system need more experimental data. Taq DNA polymerase is very sensitive to sodium dodecyl sulfate (SDS), and some non-ionic detergents can completely eliminate the inhibitory effect of low concentration SDS on enzyme activity. For example, 0.5% Twen20 and 0.5% NP40 can offset the effect of 0.01% SDS on enzyme activity.

Section 4.4 of this chapter will discuss in detail the polymerase chain reaction technology (Polymerase Chain Reaction, PCR) based on Taq DNA polymerase.

4.1.3 Ligase

DNA ligase, also known as DNA adhesive enzyme, plays a special and key role in biological computing, that is, to glue two DNAs into one, performing corresponding operations. Whether it is the adhesion of double-stranded or single-stranded DNA, DNA ligase can connect the tail end of the 3′ end of DNA with the front end of the 5′ end by forming a phosphodiester bond. Although there are other proteins in the cell, such as DNA polymerase, which can glue DNA by forming a phosphodiester bond through the polymerization reaction process when one strand of DNA is used as a template, the adhesion process of DNA polymerase is just an incidental function of the polymerization reaction. The real work of DNA adhesion in cells is mainly done by DNA ligase. In biological computing, this enzyme is mainly used for the generation of solution space and the recombination of solutions.

As the name suggests, the function of DNA ligase is to bond broken DNA, and only the reactions of DNA replication and DNA repair in cells involve the synthesis of broken DNA. Therefore, DNA ligase plays an important role in the above two mechanisms. In addition to the bonding reactions in cells, with the progress of molecular biology, almost most molecular biology laboratories will use DNA ligase

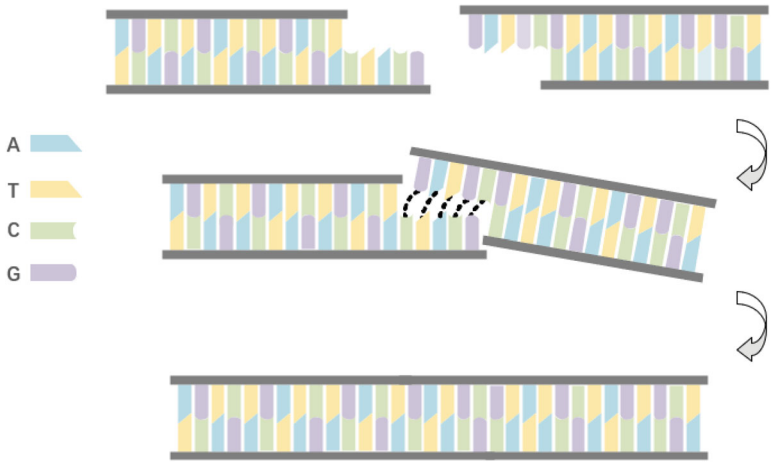


Fig. 4.3 Schematic diagram of DNA ligation process completed by ligase acting on sticky ends

to carry out recombinant DNA experiments, which may also be classified as another important function of it.

The chemical reaction process of DNA ligase, taking T4 ligase as an example, first, one end of the DNA 3′ end needs to be modified into a hydroxyl group (OH-), and the other 5′ end must carry a phosphate. The covalent bond of the phosphodiester bond is promoted by the action of DNA ligase, and the nucleotide sequence is paired in a guanine-cytosine corresponding manner to complete the reaction. DNA ligase can also handle blunt ends, that is, even if there is no guanine-cytosine paired base pair, the above reaction can still be carried out.

In prokaryotic cells and most other cells, T4 DNA ligase is the main one. In mammalian cells, at least four ligases have been found and named:

Type I Ligase It is the main DNA ligase, which connects Okazaki fragments produced during DNA replication, and relies on the function of Type I ligase in DNA recombination and repair.

Type II Ligase Type II ligase was initially purified from calf thymus and fetal calf liver, but it was later confirmed to be just a fragment cut by protease from Type III.

Type III Ligase Type III ligase forms a complex with XRCC1 protein, mainly acting on the bonding reaction of base excision repair.

Type IV Ligase Type IV ligase forms a complex with XRCC4 protein; both Type III and Type IV participate in the bonding process of DNA repair, and together participate in the last reaction of non-homologous end joining.

4.1.4 Modification Enzymes

The DNA modification enzymes commonly used in biological computation mainly include alkaline phosphatase, DNA methyltransferase, and T4 polynucleotide kinase.

Alkaline phosphatase (Alkaline phosphatase, ALP or ALKP) is a type of hydrolase that can remove phosphate groups from nucleotides, proteins, and alkaloids to perform dephosphorylation. It is most effective in an alkaline environment, hence the name alkaline phosphatase [8]. The most common application of alkaline phosphatase in the laboratory is to remove the phosphate group at the 5′ end of DNA to prevent the vector from self-circularization and to remove the phosphate group before radiolabeling the 5′ end of DNA [9].

DNA methyltransferases mainly function to methylate certain sites of DNA molecules (such as restriction enzyme recognition sequences) to protect them from being cut by restriction enzymes. In mammals, there are mainly two types of DNA methyltransferases: DNA methylation maintenance enzyme Dnmt1 and de novo DNA methyltransferases Dnmt3a, Dnmt3b, and Dnmt3L.

T4 polynucleotide kinase (Kinase) mainly catalyzes the transfer of γ-Pi of ATP to the 5′-OH of DNA or RNA, making it phosphorylated. It is commonly used in biological computation to radiolabel the 5′ end of DNA chains, i.e., to perform probe labeling.

4.1.5 Nucleases

Nucleases (nuclease) are enzymes that can cut the phosphodiester bonds between nucleic acids and nucleotides. Nucleases have different effects on the single-strand and double-strand breaks of their target molecules. Commonly used nucleases in biochemical operations include nuclease Bal 31, exonuclease III, and single-strand nuclease S1. In biological computation, this enzyme is mainly used for the elimination of non-solutions.

Nuclease Bal 3 is an enzyme produced outside the cell body of the marine bacterium A. espejiana Bal 31. It cuts from both 3′ and 5′ ends of double-stranded DNA. The enzyme starts to hydrolyze the two strands simultaneously, and the hydrolysis speeds of the two strands may not be equal. The result of the reaction is that the double strand shortens from both ends, but mostly leaves a single-strand end. The complete hydrolysis product is 5′-mononucleotide. This enzyme has highly specific single-strand deoxyribonucleic acid endonuclease activity, and can also degrade double-strand circular DNA in the single-strand area that appears momentarily in the gap or super helical curl, or gradually unravel the double-strand linear DNA at the 3′ or 5′ end. It is mainly used for deletion mutation cloning experiments of different lengths and nucleic acid structure and function analysis.

Exonuclease III is a 3′ end exonuclease, and the reaction requires Mg^{2+}. It has 3′ end phosphatase, endonuclease, and RNase H activity. It is commonly used to prepare specific probes and templates for DNA polymerase. This characteristic also makes it used in biocomputing to prepare specific probes for solution extraction.

Single-strand nuclease S1 is a single-strand (ssDNA, ssRNA) specific nuclease, producing mononucleotides or oligonucleotides with a 5′ end phosphate. It works at low pH values (pH 4–4.5) and requires Zn^{2+}. It is used to cut off the hairpin structure of DNA molecules, remove the sticky ends of DNA, and form flat ends. In biology, it is mainly used to analyze RNA-DNA hybrid structures, that is, to analyze introns. In biocomputing, it is used to eliminate non-solutions.

4.2 Biochemical Operations in Biocomputing

Biocomputing is accomplished by performing certain specific biochemical operations on biomacromolecules, mainly DNA molecules, including regulating external conditions such as temperature and pH in biochemical reactions, and artificially cutting, splicing, combining, and separating DNA molecules.

4.2.1 Synthesis of DNA Molecules

Currently, the chemical synthesis of DNA molecule fragments generally uses the solid-phase phosphoramidite method. The organic chemical principle is to use the nucleotide bound to the solid-phase carrier CPG (Controlled Pore Glass) as the first nucleotide at the 3′ end and another protected deoxynucleotide to undergo condensation reaction under the catalysis of DNA polymerase. After the 5′ end protecting group dimethylaminotriphenylmethyl (DMT) is oxidized and removed, it is condensed with the third protected deoxynucleotide. This cycle achieves the purpose of synthesizing DNA molecule fragments. After the condensation synthesis is completed, the protecting group on the DNA fragment is removed by the ammonia solution method. Currently, in DNA computing, a method called Mix-and-Split combinatorial synthesis is widely used to synthesize a large number of DNA molecule collections to represent the solution space [10]. Usually, this method uses magnetic beads as a support for DNA synthesis. It is estimated that for a magnetic bead with a diameter of 20 μm, about 6.02–10^{11} DNA molecules can be fixed, and the number of such magnetic beads in 1 gram is about 2.4–10^{8}. The synthesis starts from the 3′ end of the DNA molecule. Because the support itself is magnetic, it can be separated from the DNA molecules not on the magnetic beads in the solution through a magnetic field. The main synthesis steps are as follows: first, the magnetic beads in the U-shaped container are evenly distributed to two synthesis devices; then, the corresponding DNA sequences are synthesized in these two devices at the

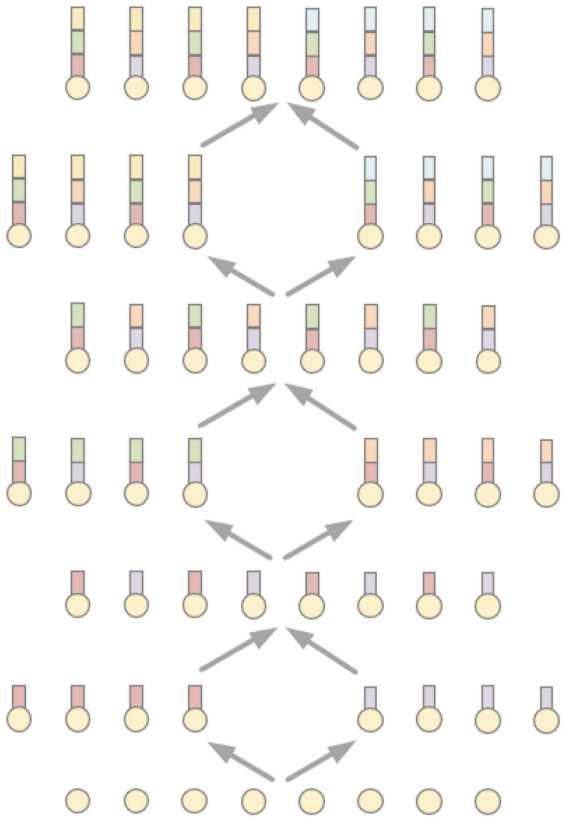


Fig. 4.4 Schematic diagram of the Mix-and-Split combinatorial synthesis method

same time; finally, the magnetic beads in these two synthesis devices are merged into the U-shaped container, and this method is repeated until all the combinatorial DNA molecules are synthesized. Figure 4.4 shows a schematic diagram of synthesizing 2^3 combinatorial DNA molecules, so this method can synthesize a large number of exponential combinatorial DNA molecules in linear time [10].

4.2.2 Cutting, Connecting, and Pasting of DNA Molecules

Operations on DNA sequences require the catalysis of the above-mentioned tool enzymes. In DNA computing, the main operations performed on DNA molecules are cutting, connecting, and pasting to complete the corresponding calculations. The commonly used ones are three enzymes: restriction endonucleases, DNA ligase, exonuclease.

The previous section introduced the tool enzymes used in biochemical operations, and the following will specifically introduce the common biological operations involved in DNA computing [11–13].

Restriction endonucleases are the scalpels of molecular biology. They are a type of DNA hydrolase that recognizes a specific position (i.e., recognition site) in a specific nucleotide sequence in double-stranded DNA, and cuts the DNA chain by hydrolyzing the $3' \rightarrow 5'$ phosphodiester bond, producing $5'$-phosphate and $3'$-OH terminal.

The recognition sequence length of restriction endonucleases is generally 4 to 6 nucleotides and most have a dyad symmetry structure, also known as a palindromic sequence. A few enzymes can also recognize longer nucleotide sequences. Restriction enzymes not only have specific recognition sequences, but any one enzyme cuts During the DNA chain, the $3'$ phosphate ester bond of the $3'$ and $5'$ diphosphate ester bond of the nucleotide is always hydrolyzed, so that the $5'$ end of the product carries a phosphate monoester group, and the $3'$ end is a free hydroxyl group. Therefore, the ends of all products of a certain enzyme have the same structure. During the calculation process, the enzyme recognizes specific sequences and performs cutting, completing one step of the calculation process.

Based on the structural characteristics of the cutting point sequence, the ends of the products can be divided into sticky ends (Sticky End) and blunt ends (Blunt End). Sticky ends refer to the single-strand structure of 1 to 4 nucleotide residues at the end of the DNA fragment after enzyme cutting, and the protruding single strands at both ends of the fragment are complementary. The protruding single strands can be divided into $5'$ and $3'$ sticky ends due to different positions. The protruding single strand with a $5'$ phosphate monoester is called a $5'$ sticky end, and the protruding single strand containing a $3'$ hydroxyl group is called a $3'$ sticky end. Blunt ends refer to the structure of the fragment after enzyme cutting. In DNA in vitro recombination, sticky ends are effective substrates for DNA ligase, with high connection efficiency.

Exonucleases are a type of enzyme that can continuously cut off nucleotides from the ends of linear deoxyribonucleic acid. Some exonucleases can remove nucleotides from the $5'$ hydroxyl end, while others can remove nucleotides from the $3'$ hydroxyl end. Some exonucleases can remove nucleotides from both the $5'$ hydroxyl end and the $3'$ hydroxyl end, such as DNA polymerase I. At the same time, some exonucleases, such as E. coli exonuclease I (Eco I) and exonuclease VII (Eco VII), are specifically targeted at single-strand molecules, while others are specifically targeted at double-strand molecules such as E. coli exonuclease III (Eco III), and some can degrade both single and double strands. Because exonucleases shorten DNA molecules by removing some nucleotides from the end of the DNA molecule each time, and can degrade the entire DNA molecule, we call it destruction (Destroy). Figure 4.5 is a schematic diagram of exonuclease, in which E. coli exonuclease III is a $3' \rightarrow 5'$ nuclease (degrading $3' \rightarrow 5'$ oriented chains), and a $5'$ protruding molecule is obtained by the action of exonuclease III. And Bal31, removes nucleotides from both strands of the double-strand molecule. In fact, many polymerases also have exonuclease behavior, which is very critical in the

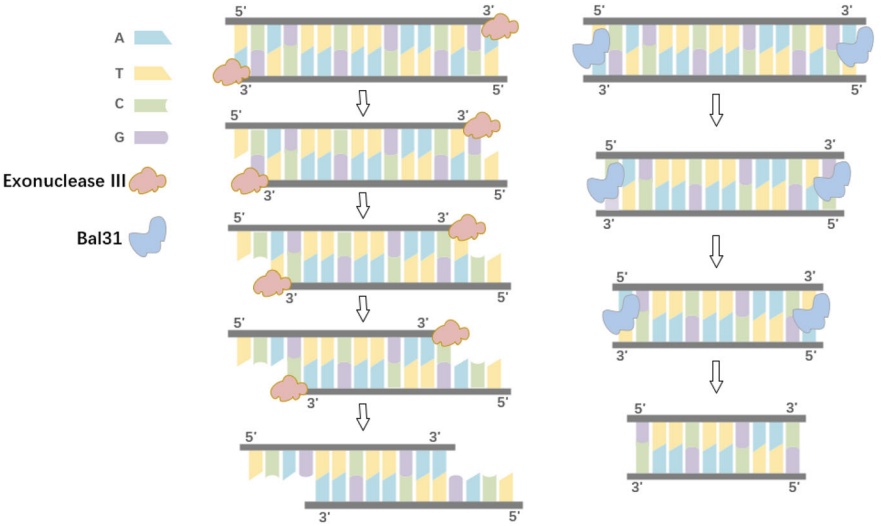


Fig. 4.5 Schematic diagram of exonuclease cutting

error correction in the DNA replication process (executed by polymerase). But polymerases are always $5' \rightarrow 3'$ oriented, and the associated exonucleases can be either $5' \rightarrow 3'$ oriented or $3' \rightarrow 5'$ oriented. During the calculation process, select specific exonucleases to eliminate the destruction of some sequences, selectively eliminate some non-solutions, and prepare for the next step.

DNA ligases are enzymes that seal gaps in the DNA chain. They catalyze the formation of a diphosphate bond between the $5'$-PO_4 of the DNA chain and the $3'$-OH of another DNA chain with the energy provided by the hydrolysis of ATP or NAD. However, these two chains must be paired with the same complementary chain (except for T4 DNA ligase), and they must be two adjacent DNA chains to be catalyzed by DNA ligase into a diphosphate bond. This form of connection process is very necessary for normal DNA replication, repair of damaged DNA, and splicing of DNA chains in genetic recombination. The most commonly used T4 DNA ligase (T4 DNA ligase) catalyzes the formation of a diphosphate bond between the adjacent $3'$-OH and $5'$ phosphate group in double-stranded DNA. It can be used to connect two DNA fragments with sticky ends, or to connect two blunt-ended DNA fragments, making them a recombinant DNA molecule. It should be noted that DNA ligase cannot connect two single-stranded DNA molecules or circular DNA single-stranded molecules. The connected DNA chain must be part of a double helix DNA molecule. In DNA computation, this enzyme is mainly used for the generation of solution space and the recombination of solutions.

4.2.3 DNA Recombination Technology

DNA recombination (DNA recombination) or molecular cloning is a process of forming a recombinant DNA molecule through a vector molecule (such as plasmid, bacteriophage, virus, etc.) with replication capability, introducing it into a recipient cell that does not have this recombinant molecule, and performing persistent and stable replication and expression, so that the recipient cell produces exogenous DNA or its protein molecule. There are four commonly used vectors, namely plasmids, bacteriophages, cosmid, and retroviruses. These vectors have characteristics such as small molecular weight, multiple enzyme cutting sites, various selective markers, stable existence in cells and independent replication and amplification capabilities. Among them, plasmid vectors were first used in DNA computation.

Plasmids are extrachromosomal molecules that can replicate independently and stably inherit a type of circular double-stranded DNA molecule (Covalently Closed Circular DNA, cccDNA). In the encoding of plasmid DNA, all plasmid DNA molecules suitable for use as gene cloning vectors must contain the following three common components: replicon structure, selective markers, and cloning sites. The replicon structure includes a replication origin (Origin, abbreviated as ori), regulatory genes controlling the replication frequency, and some replicon-encoded genes. The multiple cloning sites (Multiple Cloning Sites) of the plasmid and its circular structure can form a special data structure, which can be used for DNA computation. Plasmid-based DNA computation was first proposed by Head [14] and others, which fully utilizes the operation methods and tool enzymes of genetic engineering to solve problems. The methods for plasmid DNA recombination mainly include sticky end connection and blunt end connection. The adoption of these methods mainly depends on the nature of the ends of the exogenous DNA fragments and the nature of the restriction enzyme cutting sites on the plasmid vector and exogenous DNA.

4.2.4 Denaturation and Hybridization

Denaturation (Melting/Denaturation) and hybridization (Annealing/Hybridization) are two of the most basic operations in DNA computation, and almost all computation models use this operation, which also runs through other operations. Usually, heat denaturation is used, that is, when heated to a certain temperature (85 °C $\sim$ 95 °C), the hydrogen bonds of the DNA double helix break, forming a single strand; the hybridization process is the opposite, when the temperature drops to a certain degree, two complementary single strands will hybridize to form a double strand, so the hybridization process is also called the renaturation process.

4.2.5 Amplification of DNA Molecules

In DNA computation, it is often necessary to amplify specific DNA molecules or calculation results in order to detect the calculation results. The Polymerase Chain Reaction (PCR) technology, designed by Kary Mullis in 1985, is a rapid method for in vitro enzyme-promoted amplification of specific DNA fragments. The principle of PCR technology mainly depends on the characteristics of the DNA template, using DNA polymerase; the function of DNA polymerase is the replication and promotion of DNA synthesis; mimicking the replication process in the body in vitro, inducing polymerase reaction between the additional pair of primers. The whole process of PCR is composed of DNA template denaturation, template and primer renaturation, and primer extension, specifically described as follows:

1. Denaturation, first all target DNA double strands that need to be amplified are heat denatured into two oligonucleotide single strands;
2. Annealing, then a pair of oligonucleotide fragments complementary to the sequences near the two ends of the DNA to be amplified, synthesized artificially according to the known DNA sequence, are added as primers, i.e., left and right primers. When the temperature suddenly drops, the template DNA and the primer bind complementarily according to the base pairing principle to form a hybrid chain locally, and the chance of complementarity between the template DNA double strands is less;
3. Extension, under the action of Taq DNA polymerase, using 4 types of deoxyribonucleotide triphosphates (dNTP) as raw materials, in the presence of magnesium ions, the primer is extended in the 5′ to 3′ direction, automatically synthesizing new DNA chains, making DNA replicate into double strands again. Then the second cycle of amplification begins.

The above three steps are a cycle, the primer not only plays a guiding role in the reaction, but also plays a role in specifically limiting the range of amplified DNA fragments. The newly synthesized DNA chain contains the complementary sequence of the primer and can serve as a template for the next round of polymerase reaction. This repetition causes the target DNA fragment to be amplified exponentially. Because PCR technology is particularly important in biological computation, we will discuss it in detail in a separate section (Sect. 4.4 of this chapter).

4.2.6 Separation and Extraction of DNA Molecules

4.2.6.1 Gel Electrophoresis Technology

In DNA computation, the separation of DNA molecules of specific lengths is mainly achieved through gel electrophoresis. This method is simple and fast, and can separate fragments that other methods cannot separate. After directly embedding low-concentration fluorescent dyes, the location of the DNA fragments can be

directly detected under ultraviolet light. If necessary, DNA fragments can also be recovered from the gel for various cloning operations. In addition, combined with other methods such as PCR technology, the sequence of DNA can be analyzed.

Electrophoresis refers to the migration process of charged particles under the action of an electric field, and is the main method for separating, identifying, and purifying DNA fragments. DNA molecules are a type of strong polar molecule, with an isoelectric point of pH 2 $\sim$2.5. The principle of gel electrophoresis to separate DNA molecules is: DNA molecules have charge effects and molecular sieve effects in the gel. DNA molecules carry a negative charge in a solution above the isoelectric point and move towards the positive pole in an electric field. Under a certain electric field strength, the migration speed of DNA molecules depends on the molecular sieve effect. DNA fragments with different relative molecular masses have different electrophoretic speeds, thus achieving separation. Gel electrophoresis can not only separate DNA of different molecular weights, but also separate DNA molecules of the same relative molecular weight but different conformations.

The electrophoresis process must be carried out in a supporting medium. Currently, the two commonly used supporting materials in biological operations are agarose and polyacrylamide, respectively known as agarose gel (Agarose Gel) electrophoresis and polyacrylamide gel (Polyacrylamide Gel) electrophoresis. Agarose is purified from agar, mainly a linear polysaccharide composed of D-galactose and 3,6-anhydro-L-galactose, usually used at a concentration of 1% to 3%. After heating to boiling until the solution becomes clear, it is injected into the mold and cooled at room temperature to form agarose gel. The agarose forms a relatively stable cross-linked structure through intra- and intermolecular hydrogen bonds. This cross-linked structure gives agarose gel good anti-convection properties. Polyacrylamide gel is a gel polymerized from monomeric acrylamide (CH_2 =$CHCONH_2$Acrylamide) and methylenebisacrylamide ($CH_2(NHCOHC{=}CH_2)_2$ N, N′-methylenebisacrylamide) under the action of free radical catalysts. These two media can be made into gels of various sizes, shapes, and pore sizes, and electrophoresis can be performed on different devices. When DNA molecules pass through the sieve holes formed by the gel, short DNA molecules move faster than long ones, so they can be easily distinguished. Agarose has a lower resolution than polyacrylamide, but its separation range is wide, about 200 bp $\sim$50 kb of DNA molecules. Agarose gel electrophoresis is usually performed on a horizontal device. Polyacrylamide separates small fragments (5 bp $\sim$500 bp) effectively, and can even distinguish DNA fragments differing by 1 bp. DNA fragments larger than 10000 kb can be separated by pulsed field gel electrophoresis where the direction of the electric field changes periodically. Electrophoresis technology is used in biocomputing to separate and extract solution spaces, so it is also very important, and we will discuss it in detail in a separate section (Sect. 4.3 of this chapter).

4.2.6.2 DNA Molecule Extraction Technology

The acquisition of known sequence DNA molecules can be achieved through affinity purification. For example, suppose you want to obtain a single-strand molecule chain α from a solution S. First, synthesize $\overline{\alpha}$ ($\overline{\alpha}$ is the complementary molecule or a fragment of α, called a probe), and stick $\overline{\alpha}$ on a filter, then, use this filter to filter the S solution, α binds with $\overline{\alpha}$ and stays on the filter, while other molecules are filtered out. In this way, you can get the double-strand molecules (α and $\overline{\alpha}$) attached to the filter. Finally, put the filter into a container, decompose the double-strand molecules, remove the filter, and you can get the target molecules left in the container. Alternatively, the probe can be attached to tiny magnetic beads, put them into the solution S containing the target molecules, shake the mixed liquid, and the target molecules will attach to the magnetic beads (as shown in Fig. 4.6).

4.2.7 Detection and Reading of DNA Molecules

After the DNA computation is finished, it is usually necessary to detect whether there are solutions that meet the conditions (Detecting and Reading). The following molecular biology techniques are generally used.

4.2.7.1 DNA Molecule Hybridization

Molecular hybridization (simply called hybridization, hybridization) is one of the most basic experimental techniques in DNA computation research. Its basic principle is to use the denaturation and renaturation properties of DNA molecules to form heteroduplex double-strand molecules according to the base complementarity

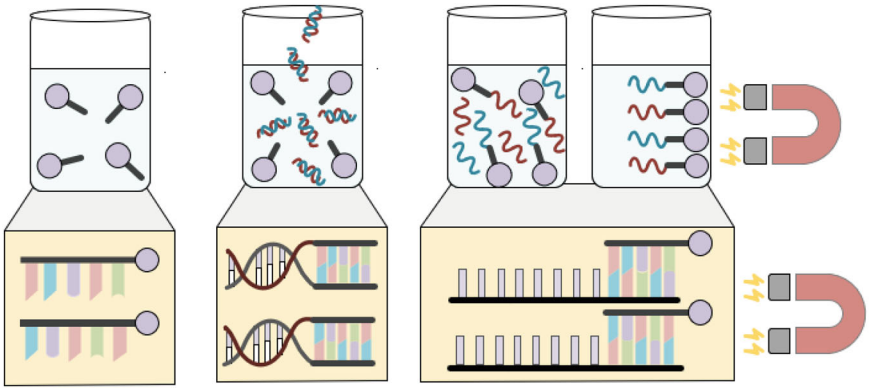


Fig. 4.6 Schematic diagram of extracting specific DNA molecules by magnetic bead method

relationship between DNA fragments from different sources. Heteroduplex double strands can form between DNA and DNA chains, RNA and DNA chains, and PNA and DNA. The essence of hybridization is to realize the renaturation of complementary nucleic acid chains under certain conditions.

Southern hybridization is a method for analyzing specific DNA sequences in vitro. During operation, nuclear DNA or mitochondrial DNA is first cut into DNA fragments with restriction endonucleases, separated by gel electrophoresis, transferred to an acetate fiber film or nylon membrane, and then hybridized with a labeled probe. Through autoradiography, it can be identified whether the special nucleotide sequence complementary to the probe exists. In DNA computation, the labeled DNA probe can also be transferred to an acetate fiber film or nylon membrane, and then hybridized with all solutions in DNA computation. After autoradiography, whether there is a solution that meets the conditions can be known by whether there is a trace.

4.2.7.2 DNA Sequence Determination

Sometimes DNA computation also needs to perform sequence analysis on DNA chains, including sequencing DNA sequences. The classic DNA sequencing method mainly uses Sanger's dideoxy chain termination method. The principle is to add four different dideoxynucleotides (formed by removing the oxygen atom on the hydroxyl group at the $3'$ end of the deoxyribonucleotide) in the extension reaction catalyzed by DNA polymerase. Because they do not have a $3'$ end hydroxyl group, they cannot form a phosphodiester bond with the subsequent four nucleotides, thereby terminating the extension of the DNA sequence. Because every base on the DNA chain has an equal chance of appearing at the variable termination end, each of the above groups of products is a mixture of oligonucleotides, the length of which is determined by the position of a specific base on the original DNA fragment. Then, under conditions that can distinguish different DNA molecules with a length difference of only one nucleotide, each group of oligonucleotides is subjected to electrophoresis analysis. As long as several groups of oligonucleotides are added to several adjacent lanes above the sequencing gel, the nucleotide sequence on the DNA can be directly read from the autoradiograph of the gel. A more advanced sequencing method is based on Sanger's dideoxy chain termination method, using fluorescently labeled primers (A, T, G, C corresponding to different fluorescent dyes) to amplify the template to be tested, and using capillary electrophoresis and excitation light detection to read the DNA sequence. Due to its high degree of automation and short time consumption, this method is adopted by many commercial sequencers.

4.2.8 Other Biochemical Operation Techniques for Biological Computation

4.2.8.1 Biochip Technology

The DNA chip was first proposed by S.P.A. Forder [15] and successfully developed in 1996 [16]. The subsequent biochip technology has become one of the most profound technological advancements since the mid-1990s, integrating microelectronics, biology, physics, chemistry, and computer science. It has significant basic research value and a clear industrialization prospect. At this stage, biochips have gradually developed from theoretical research to the initial application stage, and their types have expanded from a single DNA chip to include RNA chips, protein chips, and micro-laboratories. They have shown strong vitality and good application prospects in gene expression profile analysis, new gene discovery, gene mutation and polymorphism analysis, genome library mapping, disease diagnosis and prediction, drug screening, gene sequencing, and other research fields. This technology was rated as one of the top ten scientific and technological advancements in 1998.

A biochip refers to the use of light-guided in situ synthesis or micro-spotting methods to orderly solidify a large number of biological macromolecules such as nucleic acid fragments, polypeptide molecules, and even tissue sections, cells, and other biological samples on the surface of a support (such as a glass slide, silicon wafer, polyacrylamide gel, nylon membrane, etc.) to form a dense two-dimensional molecular array. Then it hybridizes with the target molecules in the labeled test biological sample. Through specific instruments, such as laser confocal scanning or charge-coupled device (CCD), the intensity of the hybridization signal is quickly, parallelly, and efficiently detected and analyzed, thereby judging the quantity of target molecules in the sample. Because glass slides or silicon wafers are commonly used as solid supports, and the preparation process simulates the preparation technology of computer chips, it is called biochip technology. Its biggest feature is its high-throughput parallel acquisition and processing of biological information.

4.2.8.2 Piezoelectric Gene Sensor Technology

Biosensor technology is an emerging high-tech field. Because it can provide rapid and effective analysis and detection methods, it has broad application prospects in the fields of biomedicine, environmental monitoring, food hygiene, etc. Currently, the most researched is the DNA biosensor system, among which the piezoelectric gene sensor is a new type of DNA biosensor that combines molecular biology technology, acoustics, and electronics. It is a hot spot in the research of biological gene sensors. Its main advantage is that it has high detection sensitivity, which can reach the ng level, even the pg level; and the information is intuitive, it does not need

to be labeled for application in life science research, it can not separate samples, the operation is simple, fast, and easy to connect online. Its basic working principle is based on the "inverse piezoelectric effect" of piezoelectric dielectrics, that is, when an electric field is applied in the direction of polarization of the piezoelectric dielectric, it will produce mechanical deformation; when the external electric field is removed, the deformation of the piezoelectric dielectric disappears. This phenomenon of converting electrical energy into mechanical energy is the "inverse piezoelectric effect". The transducer excites sound waves in the piezoelectric dielectric (usually quartz crystal), and uses sound waves as the means of detection. The sensor surface first fixes a single-stranded DNA probe, and then adds a solution to be tested containing a complementary DNA sequence for hybridization. The formation of double-stranded DNA structure after hybridization increases the mass of the sensor surface. Due to the sensitivity of the quartz resonator to mass changes, it affects the frequency of sound waves. Because of its above advantages, using it for DNA computation will improve the automation level of DNA computation.

4.2.8.3 Microchip Laboratory Technology

The ultimate goal of the development of biochip technology and biosensor technology is to fully integrate and automate the entire process of analysis, that is, to manufacture micro total analysis systems or microchip laboratories (laboratory-on-a-chip). The micro-laboratory (Lab-on-chip) integrates the entire process of sample preparation, biochemical reactions, and detection and analysis to form a micro-analysis system. At present, there are chip laboratories composed of heaters, micropumps, microvalve controllers, microelectrodes, electrochemistry and electroluminescence detectors, and there have been chips that integrate parts such as biochemical reactions, sample preparation, detection and analysis. For example, scientists from Nanogen, Affymetrix, the University of Pennsylvania School of Medicine, and the University of Michigan have partially integrated the three parts of sample preparation, chemical reactions, and detection by using heaters, valves, pumps, microanalyzers, electrochemical detectors or optoelectronic detectors made on the chip, and have successively made different structures of microchip laboratory prototypes [17]. For example, the biochip designed and manufactured by Gene Logic can separate DNA or RNA from the sample to be tested, and label it with fluorescence, then when the sample flows past the oligonucleotide probe fixed in the grid-like microchannel, it can capture the target nucleic acid sequence complementary to it, and use its own developed detection equipment to detect and analyze the hybridization results. Because the oligonucleotide probe on this chip has a large adsorption surface area, it can sensitively detect changes in rare genes. At the same time, because the microchannels designed by the chip have the function of concentration and enrichment, it can accelerate the hybridization reaction, shorten the test time, and thus reduce the test cost. In addition, there are also microchips that integrate the preparation of samples and the process of PCR amplification. Because the micro-laboratory has the characteristics of small size, easy to carry,

and can simultaneously parallel detect multiple biomolecules, it has a wide range of applications in the biological field. The manufacture of microchip laboratories for DNA computation is expected to build a truly fully automated DNA computer.

4.2.9 New Biochemical Operations and Techniques for Biological Computing

Here we focus on the latest CRISPR-Cas9 and its derivative gene editing technology. Given that DNA computation is a new method based on a large number of DNA molecules and using molecular biology technology for computation, in its gene route design process, it needs specifically functional components that can cooperate in cells. How to design such functional components requires the help of gene editing technology. Currently, the most widely used gene editing technologies are three, including zinc finger nucleases (effectors nucleases), TALENs technology, and the rapidly developing CRISPR-Cas9 technology. CRISPR-Cas9 and its derivative technologies are widely used in laboratories around the world due to their advantages of simple production, high efficiency, easy control, and low cost compared to other technologies.

There are many known types and components of the CRISPR-Cas system. The CRISPR system composed of Cas9 protein from Streptococcus pyogenes has attracted in-depth exploration by researchers due to its simple composition. This system only has three necessary components, namely tracrRNA, CrRNA, and Cas9 nuclease.

According to the research of Jiang and others, the basic process of the CRISPR/Cas9 system can be divided into three stages, namely the acquisition period of the spacer sequence, the expression period of CRISPR/Cas, and the DNA interference period [18].

During the spacer acquisition period, the system selectively cuts invading plasmid or bacteriophage DNA fragments. The cut fragments that meet the criteria are integrated into the CRISPR locus of the host genome to become new spacer sequences, while the existing spacer sequences in CRISPR are “records” left from previous foreign DNA invasions. Different spacer sequences are separated by repeat sequences. The selection of spacer sequences is guided by the protospacer adjacent motif (PAM) adjacent to the original spacer region, and CRISPR/Cas systems from different species have different PAMs. There are also regions encoding the Cas9 nuclease and non-coding areas that transcribe traCrRNA near the spacer and repeat sequences [19]. During the expression period of CRISPR/Cas9, the system transcribes the CRISPR array into long-chain pre-CrRNA (pre-CRISPR RNA), and also transcribes trans-activating crRNA (tracrRNA) and Cas9, which are complementary to the repeat sequences in pre-crRNA. When tracrRNA complements the region transcribed by the repeat sequences in pre-CrRNA, it forms double-stranded RNA (guide RNA), which stimulates the activity of RNA-specific nucleases such

as RNase III, cuts pre-CrRNA, and forms mature CrRNA with the participation of Cas9. Each mature CrRNA contains a guide sequence of 2 nucleotide lengths transcribed from the spacer and a region complementary to tracrRNA transcribed from the repeat, where the guide sequence can complementarily bind to the invading DNA [20]. During the DNA interference period, the complex of CrRNA and tracrRNA, guide RNA (gRNA), guides the Cas9 nuclease to find the PAM sequence on the foreign genome, and stays and recognizes at the PAM sequence. Once the spacer sequence can fully complementarily pair with the sequence on the foreign genome, the Cas9 protein will cut the invading DNA to cause breaks, thereby destroying the invading DNA to protect the bacteria and achieve the purpose of degrading foreign genetic material [21].

4.2.9.1 CRISPR/Cas9 Derived Technology

Single Base Editing Technology Based on CRISPR/dCas9

Point mutations in genes are the cause of most human genetic diseases. If they can be repaired by precise means, it may bring new treatment strategies [22]. In the presence of a homologous recombination template, point mutations can be achieved by CRISPR/Cas9, but the random insertion and deletion of bases that may be brought about by its induced non-homologous end joining (NHEJ) are potential risk factors. The Cas9 mutants Cas9n and dCas9 have no function to cut double-stranded DNA, but can play a targeting role; if there are proteins/structural domains that can catalyze specific base conversion, the single base editing technology guided by CRISPR/Cas9n/dCas9 can be constructed by referring to the design of CRISPR/dCas9-FoKI.

In 2016, David Liu's laboratory constructed a single base editor CBE (Cytosine base editors, CBE) with nCas9 protein with single cutting activity and cytosine deaminase; in 2017, it constructed ABE (Adenine base editor, ABE) single base editor. CBE and ABE can use cytosine deaminase or modified adenine deaminase to perform deamination reactions on cytosine (C) or adenine (A) within a certain range of the target site, and after DNA repair and replication, they can complete C $\rightarrow$T (G $\rightarrow$A) and A $\rightarrow$G (T $\rightarrow$C) base conversions without causing DNA double-strand breaks [23]. However, ABE and CBE single base editors cannot simultaneously perform these two types of base modifications. To solve this problem, Zhang and others fused human cytosine deaminase hAID-adenine deaminase-Cas9n (SpCas9 D10A mutant) together to form a new dual base editor A&C-BEmax. It can achieve efficient conversion of C $\rightarrow$T and A $\rightarrow$G on the target sequence, while the RNA off-target level is greatly reduced, improving the editing efficiency of C $\rightarrow$T, while the efficiency of A $\rightarrow$G is slightly reduced.

CRISPR/dCas9-FoK I Gene Editing Technology

The crRNA used by the CRISPR/Cas9 system can tolerate a certain degree of mismatch, resulting in the production of off-target effects, limiting the application of the CRISPR/Cas9 system in high-precision editing. In order to improve the accuracy

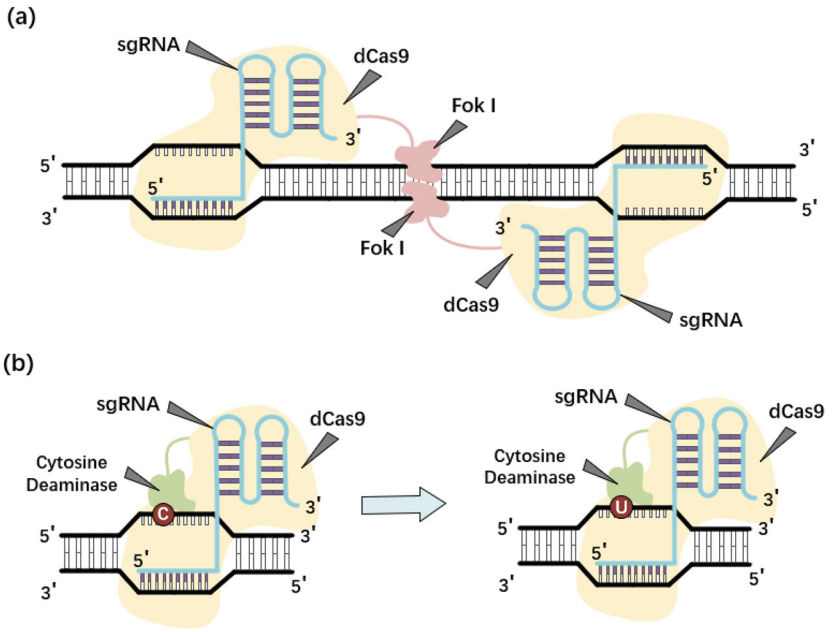


Fig. 4.7 (**a**) CRISPR/dCas9-FoK I editing technology; (**b**) Single base editing technology of CRISPR/dCas9

of the Cas9 editing system and solve the off-target problem of CRISPR/Cas9 technology, Guilinger [24] and others adopted a strategy based on dCas9. Theoretically, dCas9/sgRNA can only play a simple targeting guiding role and cannot induce DNA breaks, similar to the DNA binding domain in ZFN or TALEN. In order to achieve DNA cutting, the FoK introduced by it The cleavage structure domain of I nuclease is connected with dCas9 to form the fusion protein fCas9, which is identical to the design strategy of ZFN and TALEN. In human cell gene editing, the specificity of fCas9 is more than 140 times higher than that of wild-type Cas9. And on highly similar off-target sites, the specificity of fCas9 is at least 4 times higher than that of Cas9n. The application of fCas9 will further enrich the Cas9 toolbox and provide more comprehensive gene editing tools.

4.2.9.2 CRISPR/Cas12a Gene Editing Technology

CRISPR/Cas9 technology cannot achieve targeted sequences because the PAM sequence it recognizes needs to be rich in the base G, and the Cas9 protein has a large molecular weight, which is difficult to use in some cases. In fact, the CRISPR/Cas mechanism plays an immune defense role in many bacteria, which includes various CRISPR/Cas systems. In addition to the type II CRISPR/Cas system that uses Cas9 family nucleases as effector factors, there is another type II CRISPR/Cas system in

Prevotella and Francisella, which is also classified as a type V CRISPR/Cas system [25]. In 2015, the Zhang Feng team reported that Cpf1 (CRISPR from Prevotella and Francisella 1) (now known as "Cas12a") in the type V system is a functional bacterial immune mechanism and can mediate effective gene editing in human cells [25].

Unlike Cas9, Cas12a is guided by a CRISPR RNA (crRNA) and does not require trans-activating crRNA (tracrRNA). In addition to inducing cis-cleavage of the target DNA, binding to the target DNA also induces trans-cleavage of non-target DNA.

When recognizing a PAM sequence rich in T, Cas12a can stimulate cis-cleavage activity to cleave double-stranded DNA (dsDNA) targets. Cas12a is guided by a single crRNA, catalyzes the production of mature crRNA, and when the crRNA-Cas12a complex is formed, its conformation changes. Subsequently, this complex recognizes specific PAM sites (5′-TTTN-3′) in the non-target strand of DNA, and the RuVC structural domain cleaves the DNA target strand, producing 5′ sticky protruding ends.

It is worth noting that Cas12a's cleavage of single-stranded DNA (ssDNA) targets is PAM-independent, which is Cas12a's trans-cleavage activity. The Cas12a-crRNA-target DNA ternary complex has non-specific ssDNA trans-cleavage activity. When the complex is formed, it releases single-strand deoxyribonuclease (ssDNase) activity, indiscriminately cleaving nearby ssDNA. Literature [26, 27] uses this feature to add a single-strand RNA reporter molecule connected with a fluorescent group and a quenching group to the system, which can produce fluorescence for easy detection. Recent work using Cas12a's trans-cleavage function for biosensing includes detecting viruses, mycoplasma, single nucleotide polymorphisms (SNP), exons, crop disease diagnosis, and genetically modified organisms (GMOs) as well as small molecules.

CRISPR/Cas12a has advantages that CRISPR/Cas9 does not have, one of which is that Cas12a requires a PAM sequence rich in T bases, which helps its use in species with genomes rich in A/T bases. The Chen Jia research group at ShanghaiTech University fused the DNA-cutting-inactive dCas12a with rat-derived cytidine deaminase (APOBEC1) and found that it can effectively catalyze the conversion of C to T bases in human cells, similar to Cas9-based base editors [28]. Because it recognizes a PAM sequence rich in T bases, the Cas12a-based base editing system can complement the Cas9-based base editing system, providing more comprehensive technical conditions for related basic research and future clinical applications.

4.2.9.3 Application of Gene Editing Technology in DNA Computing

(1) Construction of Gene Editing Technology and DNA Logic Gates

The DNA computing logic gate model is one of the fastest developing and most widely used research directions in DNA computing. First, the DNA logic gate model is the underlying component to achieve DNA computing and is the foundation of

DNA computing. Basic logic gates can be cascaded to form more complex logic gates to complete more complex computing functions. In traditional electronic circuits, signals are presented at different voltage levels, with logic gate signals being true (logic high level or 1) or false (logic low level or 0). Logic gates mainly include AND, OR, XOR, NOT, NAND, NOR, and XNOR [29–32]. In 2014, a research team from the Shenzhen University Affiliated Hospital proposed a synthetic "CRISPR-Cas9-based AND gate genetic circuit" [33], which can be used to identify and regulate bladder cancer cells. This genetic circuit uses the AND gate as a model, integrates the cell information of two promoters (hTERT, hUPII) as inputs, and only activates the output gene when both inputs are active in the tested cell line. The output gene is luciferase.

(2) Biosensing Platform Based on CRISPR/Cas12a Logic Gates

In 2022, Gong's research team [34] proposed a CRISPR/Cas12a biosensing platform based on AND logic gates, which can be used for sensitive colorimetric detection of dual miRNAs. The occurrence of a disease is often accompanied by abnormal expression of different miRNAs, and the same miRNA may also be abnormally expressed in different diseases. Therefore, developing a biosensor that can simultaneously detect multiple nucleic acids can greatly improve the accuracy of disease detection.

In this study, DNA probes were designed to recognize the binary input of miRNA, with miR-944 and miR-205 as model analytes. Only in the presence of dual miRNAs, the output signal of the AND logic gate is 1, triggering the release of DNA and activating the CRISPR/Cas12a system to cleave single-stranded DNA (ssDNA) in trans. The ssDNA on the magnetic beads is cut by the activated CRISPR/Cas12a, causing the glucose oxidase (GOx) to separate from the magnetic beads, subsequently producing a colorimetric signal. The color change caused by 1 pM target miRNA can be directly distinguished by the naked eye, and the detection limit of the instrument reaches 36.4 fM. Overexpressed miR-205 and miR-944 were detected in real human serum, enabling us to distinguish between lung cancer patients and healthy individuals.

This research achievement can use a single crRNA to achieve simultaneous detection of dual miRNAs by CRISPR/Cas12a, avoiding the use of complex nucleic acid amplification and bulky equipment. The current method can expand the application of CRISPR/Cas12a for the detection of multiple biomarkers and precise disease diagnosis.

4.2.10 New Instruments Involved in Biological Computing

4.2.10.1 Atomic Force Microscope

The most direct means of detecting DNA computation results is image observation, and the self-assembly size in DNA computation is at the nanometer level, which

cannot be directly observed by traditional detection methods. Therefore, it is necessary to use advanced imaging tools, namely the Atomic Force Microscope (AFM) [35, 36].

The atomic force microscope can characterize the contour morphology of various samples with high resolution, analyze and interact with various sample surfaces, and also manipulate atoms and perform nanoscale processing using the tip. The atomic force microscope can not only see atoms and molecules, but also manipulate them, and has successively achieved the manipulation of small molecules and organic molecules. The atomic force microscope can observe and manipulate uncharged positions in the atmospheric environment, and has high practicality in the detection and manipulation of biomolecules. The atomic force microscope can be used to analyze force spectra, and the detection of sample surface morphology is based on the force between the sample surface and the probe. Therefore, the atomic force microscope inherently has the function of detecting weak forces. Now, force spectrum analysis based on atomic microscopy is the most widely used single-molecule mechanics experimental technique.

The atomic force microscope distinguishes the surface of the sample by detecting the tiny deformation caused by the interaction between the probe and the sample. The cantilever of the probe is sensitive to tiny deformations, and the tip of the cantilever is fixed with an extremely small probe, usually with a diameter of ten to several tens of nanometers. When the probe approaches the sample and contacts the sample, the two will produce interaction forces, and different forces will cause different deformations of the cantilever. This deformation is detected by a laser or

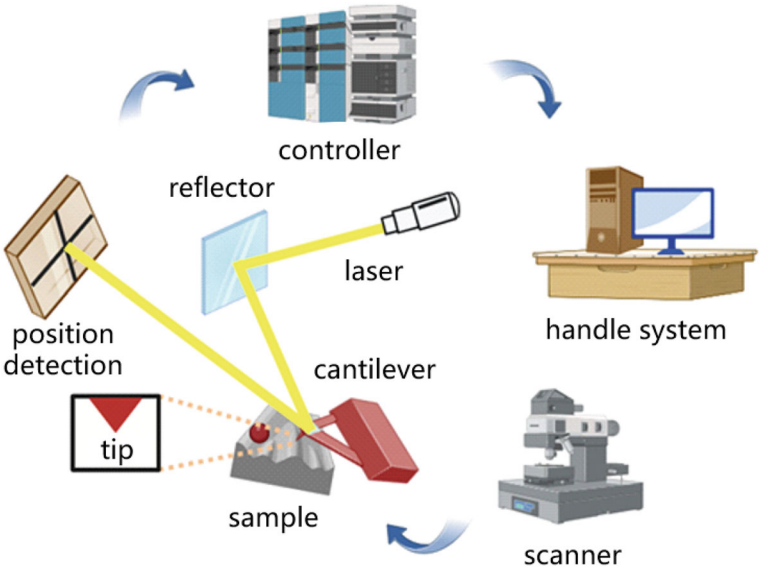


Fig. 4.8 Schematic diagram of AFM principle

tunneling current to obtain information about the sample surface. The schematic diagram of the principle is shown below.

Atomic force microscopy is divided into three working modes according to the contact mode of the probe and the sample: contact mode, tapping mode, and non-contact mode. Or it can be divided according to the working environment: gas-like mode, liquid mode, or gas-liquid mixed mode. For flexible and fragile samples, the tapping mode has a higher resolution and will not damage the sample structure. For biologically active samples, the liquid mode can keep the sample under natural environmental conditions for detection [37]. For DNA self-assembled samples, both gas-like and liquid models can be used. The sample under the gas-like mode will be lower than the sample under the liquid, and the sample under the liquid will shift position with the scanning of the probe tip. Therefore, which scanning mode to adopt depends on the comprehensive consideration of the experimenter.

4.2.10.2 Super-Resolution Fluorescence Microscope

The fluorescence microscope is affected by the wavelength of light, and can generally only reach 0.2–0.4 μm, so seeing molecules at the nanometer level is a very challenging task. Traditional characterization methods for observing nucleic acid nanostructures in a single molecule manner mainly rely on atomic force microscopy (AFM) and electron microscopy (EM). Although these methods can provide information about the assembly quality and overall structural features of nanostructures, they also have their limitations. Compared with AFM, EM is more effective in observing large sample areas, but it is insufficient for visualization due to lack of contrast under non-biological conditions. Most importantly, both methods lack chemical (chain) specificity and are limited to electron density and surface morphology maps.

Super-resolution fluorescence microscopy provides biologists and nanoscientists with important tools for studying single molecule conformations and dynamics in nanoscale biomolecules and synthetic systems [38, 39]. The progress of methods such as STED, SIM, PALM, STORM, and PAINT has enabled the optical resolution of subcellular and nanoscale structures to reach 10–20 nm.

Super-resolution fluorescence microscopy provides a method for replacing nucleic acid nanostructure microscopy, with single-chain visibility and high specificity multiplex detection, and operates in a biocompatible environment, belonging to the stochastic localization microscopy system (also known as single molecule localization microscopy or SMLM). In short, stochastic localization super-resolution visualization is achieved by randomly switching each target between the fluorescent on and off states, separating the nearby target fluorescence emissions, and using sub-diffraction limit accuracy to determine their respective positions. These methods include PALM (photoactivated localization microscopy) [40, 41], STORM (stochastic optical reconstruction microscopy) [42, 43] and PAINT (points accumulation for imaging in nanoscale topography) [44] and its various variants, achieving stochastic single molecule conversion.

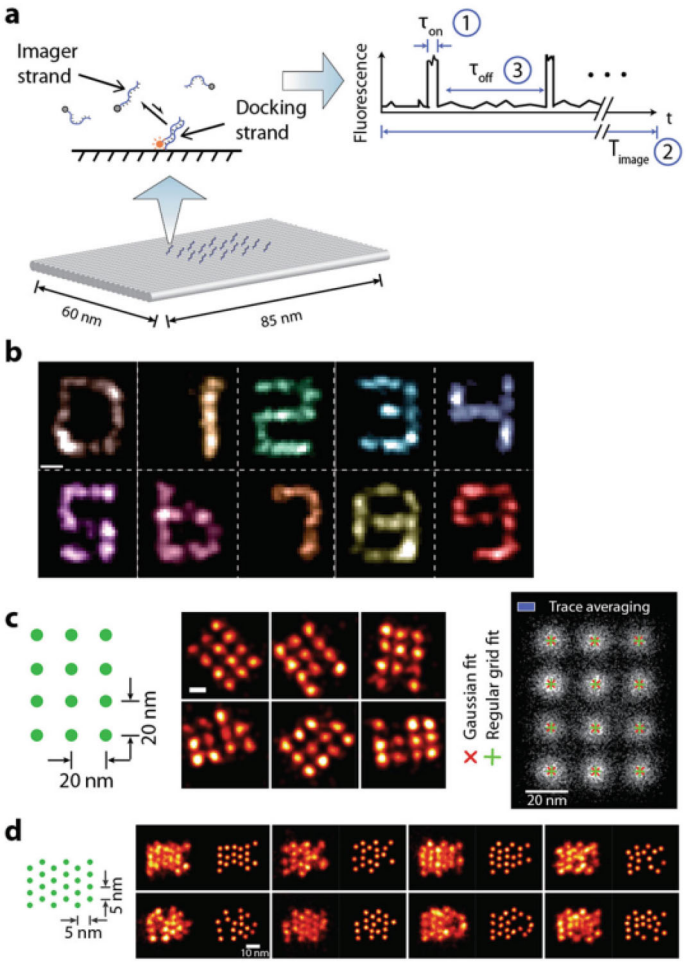


Fig. 4.9 DNA-PAINT technology

Far-field super-resolution fluorescence microscopy has allowed the observation of biomolecules and synthetic nanoscale systems with nanoscale features, and has the ability of chemical specificity and multiplexing. DNA-PAINT (points accumulation for imaging in nanoscale topography based on DNA) is a super-resolution method that uses the programmability between short oligonucleotide chains to achieve single molecule labeling and visualization resolution of at least 5–10 nm. DNA-PAINT provides a characterization method for nucleic acid nanostructures with high spatial resolution and single-chain visibility.

Methods based on the PAINT principle rely on the diffusion and random instantaneous binding of specific fluorescent groups with conjugated affinity probes. When bound to the target, the fluorescent group temporarily stays and produces

a noticeable bright flicker on the recorded camera frame. The relative brightness (or signal-to-noise ratio of the flicker) is determined by the cumulative photon emission of the bound background relative to the unbound free diffusion probe, and the signal can be enhanced by placing the sample in a total internal reflection (TIR) illumination setting. After introducing the PAINT principle, researchers have developed several variants of different affinity probes (including DNA-PAINT, uPAINT, BALM, Jungmann, etc.). Instantaneous binding between short oligonucleotide chains can be used as an affinity probe to produce a flicker pattern suitable for super-resolution imaging (DNA-PAINT). Short oligonucleotide chains ("docking chains") are labeled on molecules of interest and complementary sequences ("imaging chains"). This method has been rapidly applied to the study of nucleic acid nanostructure conformation and defects, single molecule binding dynamics, and nucleic acid substrate detection, etc. [45, 46].

4.2.10.3 DNA Synthesizer

The advancement of technology often drives theoretical innovation and breakthroughs. The theory and related experimental research of computation and storage based on DNA molecules are non-traditional computation and high-density storage methods developed on the basis of DNA synthesis and sequencing technology.

The emergence of artificially synthesized DNA is the technical basis for the realization of DNA computation. DNA synthesis is the starting point for using DNA molecules for information encoding, storage, and computation. Since 1994, when Adleman used artificially synthesized DNA molecules to encode the vertices and connecting edges of a graph, DNA computation and DNA programmable nano self-assembly technology have relied on the artificial synthesis of DNA molecules. The quality and synthesis cost of artificially synthesized DNA encoding sequences have always been one of the important factors affecting the development of DNA computation and DNA nanotechnology.

The DNA synthesizer, which serves as the starting point for DNA coding storage and computing tools, is an automated instrument for synthesizing DNA or RNA. It generally uses solid-phase synthesis methods, and the automated design of the instrument adds a specific nucleotide pre-encoded design to the oligonucleotide chain each time, extending and lengthening it. Each added nucleotide uses the same chemical reaction to interact with the corresponding purine or pyrimidine base. The biochemical reactions used vary with different instruments, and phosphoramidite method for artificial DNA synthesis is currently the most widely used method.

1. Steps and Working Principles of Automatic DNA Synthesis

The general operation steps of the DNA synthesizer are introduced using phosphoramidite oligonucleotide synthesis as an example. The reagents for phosphoramidite DNA synthesis include: DMT protecting the 5′-hydroxyl of the base, A, G, C, T phosphoramidite monomers, tetrazole coupling catalyst, acetic anhydride, N-

Fig. 4.10 Laboratory small DNA synthesizer

methylimidazole capping reagent, trichloroacetic acid (TCA) deprotection solution, 12 oxidation mixture, acetonitrile cleaning solvent, and ammonia cleavage solution. Phosphoramidite solid-phase automatic synthesis includes 4 basic steps:

The first step is to react the nucleotide with the active group pre-connected to the solid-phase carrier CPG with trichloroacetic acid, remove its 5′-hydroxyl protecting group DMT, and obtain the free 5′-hydroxyl.

The second step, the raw material for synthesizing DNA, the phosphoramidite protected nucleotide monomer, is mixed with the activator tetrazole to obtain the nucleotide phosphite activated intermediate. Its 3′ end is activated, the 5′-hydroxyl is still protected by DMT, and it undergoes condensation reaction with the free 5′-hydroxyl in the solution.

The third step is the capping reaction. In the condensation reaction, a very small number of 5′-hydroxyls may not participate in the reaction (less than 2%). They are terminated by acetic anhydride and 1-methylimidazole to prevent further reaction. These short fragments can be separated during purification.

The fourth step, under the action of the oxidant iodine, the phosphoramidite form is converted into a more stable triphosphate.

After the above four steps, a deoxynucleotide is connected to the nucleotide on the solid-phase carrier. Then, its 5′-hydroxyl protecting group DMT is removed with trichloroacetic acid, and the above steps are repeated until all the required bases are attached. The synthesis efficiency can be determined by observing the color of the TCA treatment stage during the synthesis process.

Through high-temperature treatment with ammonia, the primer connected to CPG is cut off, and the primer is purified by OPC, page and other means. The finished primer is concentrated with C18, desalted, and precipitated. The precipitated primer is suspended in water, quantified by measuring OD260, and packaged according to the order requirements.

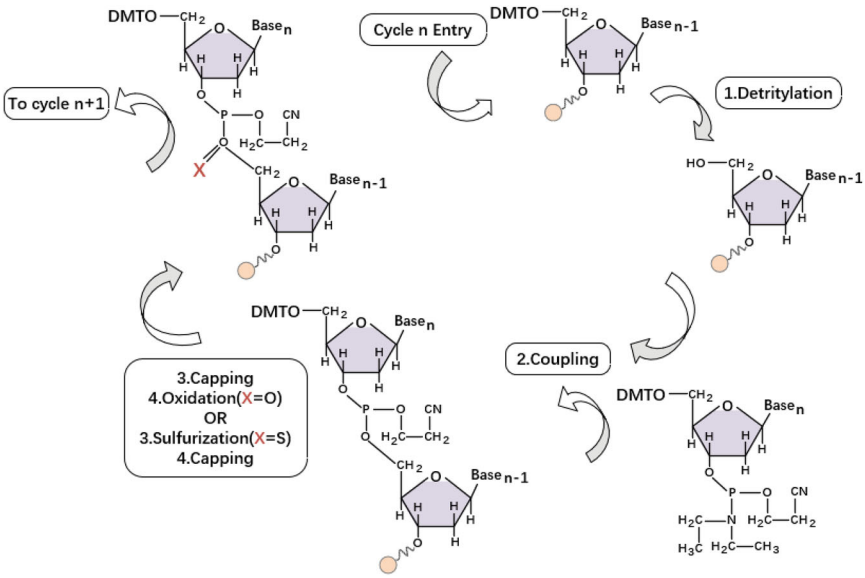


Fig. 4.11 Principle of phosphoramidite method solid phase automatic DNA synthesis

2. Chip-Type DNA Synthesizer

The main manufacturers of current DNA synthesizers are committed to the perfection of machine performance and the exploration of application fields, as well as the development and research of high-efficiency, high-yield instruments. High-throughput, high-density chip-type DNA synthesizers will be the future development direction, which can simultaneously synthesize tens of thousands of different DNAs on the DNA synthesis chip. In the chip-type DNA synthesizer, the DNA synthesis chip containing a large number of synthesis pools is placed in the synthesizer. The synthesizer controls the electrode potential of each synthesis pool in the chip. On the chip (generally 12,000 or 90,000 synthesis pools, that is, a chip that can simultaneously synthesize 12,000 or 90,000 DNAs), according to the DNA sequence designed by the software, a certain length of DNA is synthesized from the $3'$ end to the $5'$ end of the primer to be synthesized. The adjacent nucleotides are connected by a $3' \rightarrow 5'$ phosphodiester bond, thereby synthesizing the required DNA/RNA. After synthesis, a large amount of DNA is cut off from the chip, and a single tube will contain tens of thousands of artificially synthesized DNAs.

In addition to high-throughput chip synthesis, because the number of base pairs of most nucleic acids needed by people far exceeds the number of base pairs of the longest nucleic acid chain that a DNA synthesizer can synthesize, chip-type DNA synthesizers can break through the existing nucleic acid synthesis. The number of base pairs is limited, and super-long DNA chains are quickly synthesized. In addition, chip-type DNA synthesizers also have advantages such

as in-situ electrochemical synthesis on the chip, electrochemical deprotection, and fast synthesis speed, greatly improving the efficiency of artificial DNA synthesis.

4.2.10.4 DNA Sequencer

Gene sequencing is the technical foundation for large-scale DNA data storage. The rapid development of the Human Genome Project and the resulting gene sequencing technology is the prerequisite for the realization of large-scale DNA data storage. Currently, the cost of DNA sequencing has been reduced to about one cent per base, which greatly promotes the development of theories and technologies related to DNA molecular storage. At the same time, the third-generation nanopore DNA sequencer can directly convert the encoded information carried by a large number of DNA molecules into photoelectric signals through specially designed nanopores, so it has high throughput and highly parallel data reading characteristics, and may become the universal output device for future DNA molecular computers.

1. Basic Working Principle of DNA Sequencer

The working principle of the current DNA sequencer is mainly based on the double deoxy chain termination method invented by Sanger or the chemical degradation method invented by Maxam-Gilbert. Although these two methods are different in principle, they both start the extension of the nucleotide chain at a fixed site, randomly terminate at a specific base, and produce four groups of nucleotide chains of different lengths with A, T, C, and G at the end. The DNA sequence is obtained by separating and detecting the fragments on the denatured polyacrylamide gel by electrophoresis. Because the double deoxy chain termination method is simpler and more suitable for optical automatic detection, it is widely used in fully automatic DNA sequencers that purely aim to determine the DNA sequence. The chemical degradation method has important application value in studying the secondary structure of DNA and the interaction between protein and DNA. Here, the sequencing principle of the double deoxy chain termination method is mainly introduced. The specific principle is described in Sect. 4.2.7.2, and the schematic diagram is shown in Fig. 4.12.

2. Development of DNA Sequencer

First-Generation DNA Sequencer

The first-generation DNA sequencer is based on the classic double deoxy nucleotide termination sequencing method proposed by Sanger and others. Subsequently, in the mid-1980s, automatic sequencers appeared that replaced radioactive isotope labeling with fluorescent labeling and replaced radioactive autoradiography with fluorescent signal receivers and computer signal analysis systems. Later, the capillary electrophoresis technology that appeared in the mid-1990s greatly increased the throughput of sequencing. The traditional first-generation sequencing technology has the advantages of high accuracy and simplicity and speed, but due to the low sequencing throughput, it is only suitable for the identification of small sample genetic disease genes, and it is difficult to

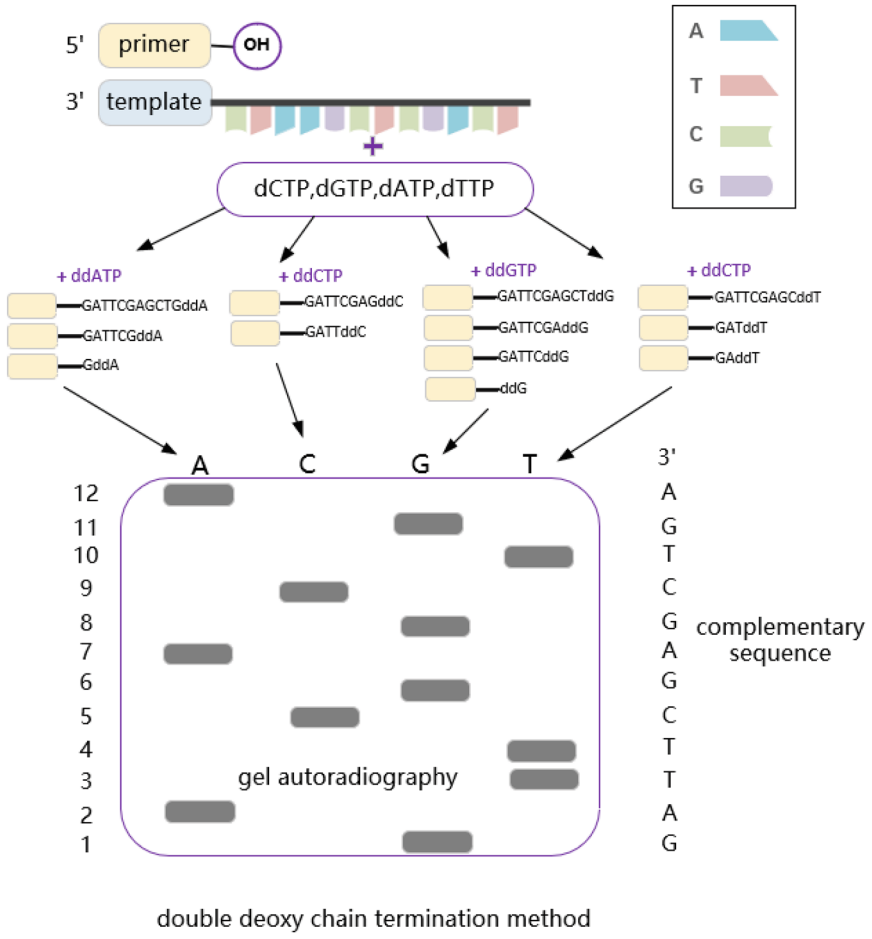


Fig. 4.12 Principle of DNA sequencing

complete the screening of large sample cases without clear candidate genes or a large number of candidate genes.

Second-Generation DNA Sequencer

The second-generation DNA sequencer uses the second-generation sequencing technology developed in this century. By connecting the ends of the fragmented genome DNA to be sequenced with adapters, millions of spatially fixed PCR clone arrays are produced by different methods. Then carry out primer hybridization and enzyme extension reactions. Complete DNA sequence information is obtained through computer analysis. DNA sequencers using second-generation sequencing technology not only maintain high accuracy, but also greatly reduce sequencing costs and greatly increase sequencing speed. The most notable feature of the second-generation sequencing technology is high throughput,

which can sequence hundreds of thousands to millions of DNA molecules at a time, making it convenient and easy to sequence the transcriptome or deep sequencing of the genome of a species.

Third-Generation DNA Sequencer

The notable feature of the third-generation sequencer is long read length, which can significantly increase the sequencing read length by 10–50 times while maintaining high accuracy. The sequencing technology used by the third-generation sequencer mainly solves the PCR amplification process required for the preparation of the sequencing library by the second-generation sequencer, to a certain extent eliminates the systematic errors introduced by the PCR amplification process, and also reduces the running time required for overall sequencing.

Fourth-Generation DNA Sequencer

The sequencing technology used by the fourth-generation DNA sequencer also belongs to single-molecule sequencing, but its principle of using nanopore chip to detect single-molecule sequencing signals no longer depends on high-speed cameras or high-resolution CCD cameras, which greatly reduces the cost of detection equipment. The basic principle of most nanopore sequencing technologies is to detect the affected current or light signal when a DNA molecule passes through a hole. Because it has a qualitative leap compared with the third-generation sequencing technology, it is usually called the fourth-generation sequencing technology.

4.3 Key Technology of Biological Computing: Gel Electrophoresis

Electrophoresis is a type of separation identification technology designed based on the physical and chemical properties of samples. It separates samples by the difference in the ability of samples to move in conductive media under the action of an electric field [47]. Generally, cations migrate towards the negatively charged cathode, and cations with a larger charge ratio migrate faster than those with a smaller charge ratio. Anions migrate towards the positively charged anode, while neutral ions remain stationary and are not affected by the electric field. In 1807, Professors Strakhov and Roys of Moscow University first observed the electrophoretic phenomenon, noting that the introduction of a constant electric field would cause the migration of clay particles dispersed in water [48]. Electrophoresis is widely used in laboratories for the analysis of large biological molecules such as DNA, RNA, and proteins. Based on the medium system and physicochemical characteristics, it mainly includes gel electrophoresis, immunoelectrophoresis, capillary electrophoresis, dielectrophoresis, and isoelectric focusing, etc. These electrophoretic techniques are used in bioinformatics for the separation, extraction, and purification of solutions.

4.3.1 Basic Principles

According to the double-layer theory related to the electrophoretic particle microsystem, the suspended particles to be separated have surface charges. An external electric field applies a static Coulomb force to them, which is affected by the microenvironment of the particle surface (Fig. 4.13). Generally, the surface charges of the particles to be separated in the electrophoretic system are shielded by the ion diffusion layer, which has a consistent charge property, but opposite to the charge property of the particle surface. The electric field applies a static Coulomb force to the particles to be separated wrapped in the diffusion layer, and its direction is opposite to the force acting on the diffusion layer. The latter force is not actually acting on the particles to be separated, but on the ions in the diffusion layer at a certain distance from the particle surface, and is transmitted to the particle surface through viscous stress, manifesting as electrophoretic resistance. When an electric field is applied, the charged particles to be separated move stably in the diffusion layer, and the total force is zero [49].

Considering the viscosity of the diffusion layer on the resistance of the moving particles, under a certain electric field strength E, the drift velocity of the suspended particles v is proportional to the external electric field, then its electrophoretic migration rate μ_e is defined as:

$$\mu_e = \frac{v}{E}$$

The most famous electrophoresis theory was proposed by Smoluchowski in 1903 [50]:

$$\mu_e = \frac{\varepsilon_r \varepsilon_0 \zeta}{\eta}$$

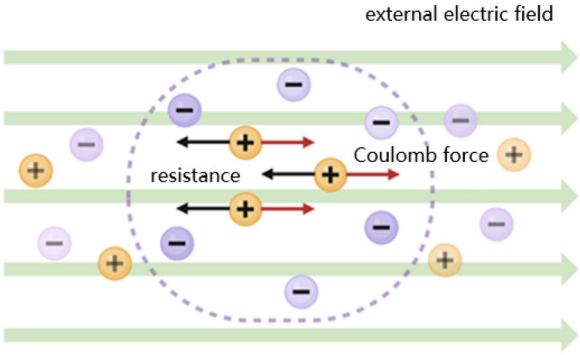


Fig. 4.13 Schematic diagram of electrophoresis principle

Where, ε_r is the dielectric constant of the diffusion system, ε_0 is the vacuum dielectric constant, η is the dynamic viscosity of the diffusion medium, ζ is the Zeta potential.

4.3.2 Gel Electrophoresis

Gel electrophoresis is a common method for separating and identifying large biological molecules (DNA, RNA, and proteins). The migration speed of samples on the gel carrier is directly related to the molecular size and surface charge, and the electric field that drives the sample movement is composed of the positive and negative poles of the electrophoresis instrument. The sample is placed in the sample hole of the gel material, the gel as a whole is placed in the electrophoresis chamber, and connected to a constant voltage source (Fig. 4.14). When an electric field is applied, larger molecules move relatively slowly in the gel, while smaller molecules move faster, resulting in different bands of large biological molecules with different physicochemical properties on the gel for subsequent analysis operations.

The most commonly used types of gels are agarose gels and polyacrylamide gels. Each type of gel is suitable for different types and sizes of samples depending on the preparation method. Polyacrylamide gels have a high resolution for small DNA fragments (5–500 bp), agarose gels have a relatively low resolution for DNA, but a larger separation range, usually used for DNA fragments of 50–20,000 bp, and the resolution of pulsed field gel electrophoresis may exceed 6 Mb. Polyacrylamide gels are electrophoresed in a vertical placement mode, while agarose gels are usually electrophoresed in a horizontal placement mode. The two have different preparation methods, agarose is thermally cured, and polyacrylamide is formed by a chemical polymerization reaction.

In addition to the common analysis of nucleic acid samples, gel electrophoresis can also be used for protein analysis and identification. Polyacrylamide gel electrophoresis has both charge effect and molecular sieve effect, which can separate proteins with the same molecular size but different numbers of charges. Further, through two-dimensional electrophoresis technology, proteins with the

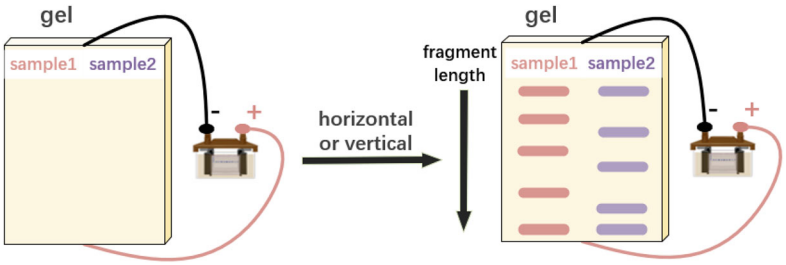


Fig. 4.14 Schematic diagram of gel electrophoresis

same number of charges but different molecular sizes can be separated. Based on the addition of antigen-antibody reactions, agarose immunoelectrophoresis can be used for the purity identification of protein preparations, the composition analysis of protein mixtures, serological characteristics, and other systematic studies.

4.3.3 Immunoelectrophoresis

Immunoelectrophoresis is a general term for a series of electrophoresis methods for separating and characterizing proteins based on antigen-antibody reactions [51]. In the electrophoresis system, the antibody reacts with the protein to be identified through non-covalent bond chemical reactions (Fig. 4.15).

Immunoelectrophoresis developed and was widely used in the second half of the twentieth century, promoting the development of biology and medicine. According to the physicochemical characteristics of the electrophoresis system, it is divided into: one-dimensional immunoelectrophoresis, two-dimensional quantitative immunoelectrophoresis, rocket immunoelectrophoresis, fusion rocket immunoelectrophoresis, and affinity immunoelectrophoresis.

Although immunoelectrophoresis has obvious advantages, two factors limit its further development: (1) the labor-intensive work in the laboratory requires a large number of experienced experimental technicians to participate, limiting large-scale automation; (2) the technical principle requires a large amount of antibody use, which raises the barrier to the application of the technology.

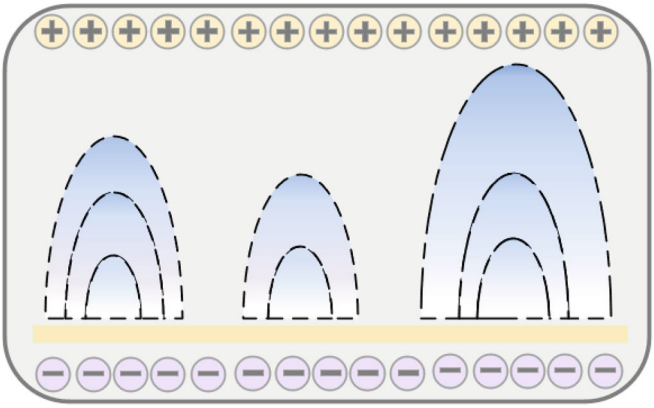


Fig. 4.15 Schematic diagram of immunoelectrophoresis

4.3.4 *Capillary Electrophoresis*

Capillary electrophoresis (CE) is a series of electrophoresis separation techniques performed in capillaries and micro-nano fluid channels with sub-millimeter diameters [52]. Generally, CE refers to capillary zone electrophoresis, capillary gel electrophoresis, capillary isoelectric focusing, capillary isotachophoresis, and micellar electrokinetic chromatography. In CE methods, the sample migrates through the electrolyte solution under the influence of an electric field, and separation is carried out according to ion migration rate, non-covalent interactions, etc. [53]. In addition, samples can be concentrated or aggregated through conductivity and pH gradients to achieve further efficient separation and identification.

The equipment required for capillary electrophoresis is relatively simple (Fig. 4.16). The main components of this system are the sample pool, source bottle, capillary path, positive and negative electrodes, high-voltage power supply, high-sensitivity detector, laser probe, and data processing device. The sample bottle, source bottle, and capillary are filled with electrolyte, such as aqueous buffer solution. To introduce the sample, the capillary inlet is inserted into the sample pool, and the sample is introduced into the capillary by capillary siphon or electrodynamic pressure, and communicates with the source bottle. The sample migration is driven by the electric field applied between the source bottle and the sample bottle.

The high-voltage power supply device provides power. In common CE instruments, the sample is separated due to its electrophoretic migration and detected near the capillary outlet. The detector signal is sent to the processing device, and the data is displayed as an electrophoretic peak map. Although the sample volume in CE is very small (usually only a few nanoliters of liquid are introduced into the capillary), the injection strategy causes the analyte concentration to be concentrated in the

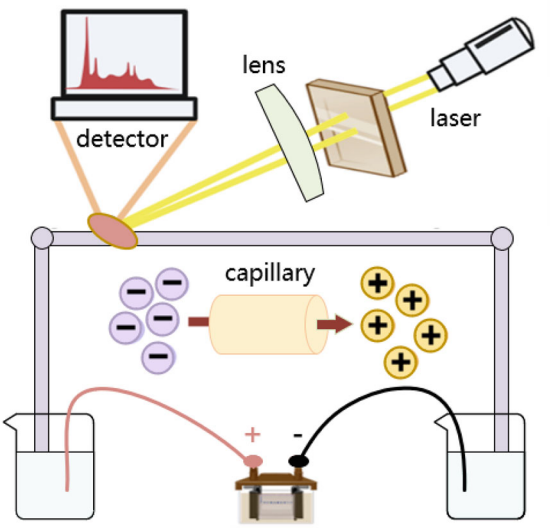


Fig. 4.16 Schematic diagram of capillary electrophoresis

capillary, enabling high-sensitivity detection. With the advancement of technology, parallel capillary arrays have emerged to achieve larger sample processing volumes. Array electrophoresis with 96 capillaries can be used for high-throughput capillary DNA sequencing. The capillary array inlet accepts samples from a standard 96-well plate, and other basic principles are similar to those shown in Fig. 4.16 [54]. Capillary electrophoresis has become an important and cost-effective method for DNA sequencing, which can currently provide high-throughput and high-accuracy sequencing information, and the sequencing speed is greatly guaranteed [55].

In recent years, the developed affinity capillary electrophoresis is a special type of capillary electrophoresis, which uses intermolecular interactions to understand the interaction information between proteins and ligands, and has a wide range of applications. Ren and others introduced new high-affinity interactions from the hydrophobic and polar interactions between IL-1α and the aptamer by adding modified nucleotides to the aptamer [56]. Huang and other researchers studied protein-protein interactions, using 6-carboxyfluorescein-labeled α-thrombin binding aptamer as a selective fluorescent probe, and studied the binding site information of protein-protein and protein-DNA interactions [57]. Affinity capillary electrophoresis has simple, fast, and low sample demand analysis characteristics, provides in-depth and efficient details for sample ligand identification, sample separation and detection, and has been proven to have high practicality in life science research.

4.3.5 Dielectrophoresis

Dielectrophoresis (DEP) is an electrophoretic technique that applies force to dielectric particles under the action of a non-uniform electric field [58]. All sample particles in the diffusion system show dielectrophoretic activity under the action of an electric field [59]. The intensity of the force largely depends on the electrical properties of the medium and particles, the shape and size of the particles, and the frequency of the electric field (Fig. 4.17). An externally applied electric field of a specific frequency can flexibly manipulate sample particles, used for cell separation, directional control of nanomaterials, and other research work [60].

The simplest theoretical model describes the sample to be separated as a uniform sphere surrounded by a conductive medium. For a uniform sphere with a radius of r, and a complex dielectric constant of ε_p^*, in a medium with a complex dielectric constant of ε_m^*, the force applied under time-averaged conditions is [59]:

$$< F_{DEP} >= 2\pi r^3 \varepsilon_m Re\{\frac{\varepsilon_p^* - \varepsilon_m^*}{\varepsilon_p^* + 2\varepsilon_m^*}\}\nabla|\overline{E_{rms}}|^2$$

Specifically, when polarized particles are suspended in a non-uniform electric field, dielectrophoretic behavior occurs. The electric field polarizes the particles, and

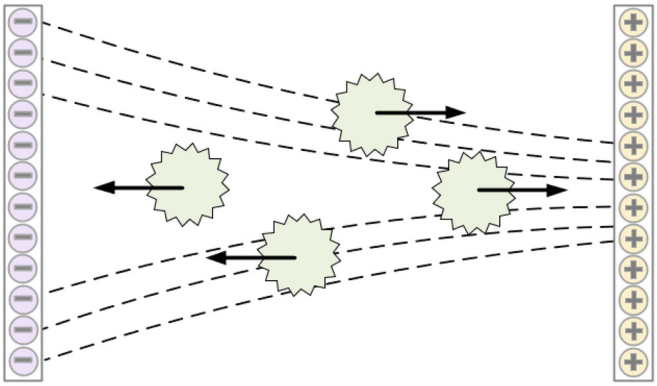


Fig. 4.17 Schematic diagram of dielectrophoresis

then the two poles along the electric field lines are subjected to a force, which can be attractive or repulsive depending on the direction of the dipole. Because the electric field is not uniform, the pole subjected to the maximum electric field will overwhelm the other pole, and the particle will move. The orientation of the dipole depends on the relative polarization rate of the particle and the medium, which complies with the Maxwell-Wagner-Sillars polarization. Since the direction of the force depends on the gradient of the electric field rather than the direction of the electric field, DEP occurs in both AC and DC electric fields. Since the relative polarization rate of the particle and the medium is frequency-related, changing the excitation signal and the way of measuring force changes can be used to determine the electrical properties of the particles.

DEP can be used to separate particles with different polarization rates, as they move in different directions under a given AC electric field frequency. Based on the understanding that biological cells have dielectric properties, DEP has many applications in medicine, including cancer cell separation and platelet separation, medical diagnosis, drug discovery, cell therapy, and other fields [61–63]. DEP has been used for the separation of live and dead cells, and the remaining live cells after separation still have vitality or are used for forced contact between selected single cells to study cell-cell interactions [64]. DEP is also combined with semiconductor chip technology to develop DEPArray technology, which can manage thousands of cells in microfluidic devices [65].

4.3.6 Isotachophoresis

Isotachophoresis (ITP) is a technique in analytical chemistry used for the selective separation and concentration of ionic analytes [66]. Charged samples are separated according to ion mobility, where ion mobility refers to the speed at which ions

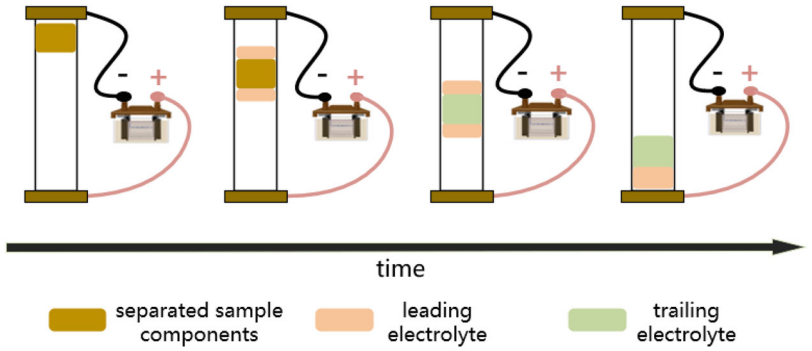


Fig. 4.18 Schematic diagram of isoelectric focusing

migrate in an electric field. The classic ITP separation technique uses a discontinuous buffer system, and the sample is introduced between the fast electrolyte area and the slow electrolyte area. The mobility of the leading fast electrolyte is higher than that of the sample components, and the mobility of the trailing slow electrolyte is lower than that of the sample components. The separated sample components are sandwiched in the middle according to their different mobilities. Under the action of a strong electric field, the separated sample components move in the gap between the leading electrolyte and the trailing electrolyte, ultimately achieving separation (Fig. 4.18). The currently popular form of ITP is transient ITP, which alleviates the limitations of traditional ITP's limited separation capacity due to overlapping analyte bands. In transient ITP, the analytes are first concentrated by ITP, and then can be separated by zone electrophoresis. Transient ITP has a wider range of applications than traditional ITP, and can easily serve as a pre-enrichment step in capillary electrophoresis (CE) separation, making CE more sensitive and achieving better separation efficiency [67].

4.4 Key Technology in Biological Computing: Polymerase Chain Reaction

Polymerase Chain Reaction (PCR) is a molecular biology technique based on the mechanism of DNA molecule replication. It achieves rapid amplification of DNA sequences under thermal cycling conditions through specific primers and DNA polymerase [4]. During the PCR process, the DNA template denatures into a single strand at high temperatures, then binds to the target sequence at a lower temperature through specific primers, and finally, new chain synthesis is carried out by DNA polymerase at an appropriate temperature. Primers with higher affinity bind to the target sequence faster, thereby rapidly amplifying the target DNA fragment after several thermal cycles. In 1983, American biochemist Kary Mullis first invented the

PCR technique. He found that by simulating the DNA replication mechanism in organisms, rapid amplification of DNA can be achieved under laboratory conditions [68]. PCR technology is widely used in the laboratory for DNA amplification and analysis, and in bioinformatics for amplifying and extracting solution spaces and implementing "amplification" operations. Depending on the purpose of the experiment and technical characteristics, it mainly includes quantitative PCR (qPCR), reverse transcription PCR (RT-PCR), digital PCR (dPCR), and multiplex PCR.

4.4.1 The Journey of PCR Invention

On a hot afternoon in May 1983, Kary Mullis drove his silver Honda from Berkeley, crossed over Coverdale, and headed towards Anderson Canyon. The branches of the California buckeye stretched over Highway 128, the pink and white branches looked particularly cold in the sunset, and the fragrance of the flowers filled the warm air. It was a night full of buckeye fragrance, but Mullis had a vague restlessness in his heart. His thoughts flew back to the lab, images of DNA chains curling and floating in his mind. He was busy with his favorite pastime—thinking about how to read the sequence of the king of molecules, DNA. The complexity of DNA and the infinite possibilities behind it fascinated Mullis. He realized that if he could decipher the blueprint of DNA, many genetic defects and disease tragedies could be predicted and avoided. An idea suddenly flashed in Mullis's mind: if he could design a short synthetic DNA fragment to recognize a specific sequence, and then start a program that allows the sequence to continuously replicate itself, he could solve his problem. This idea, although simple, was revolutionary. One of the instincts of DNA molecules is to replicate themselves, and Mullis wanted to take advantage of this natural characteristic.

That night, Mullis stopped the car and hastily sketched his idea on an envelope with a pencil. His girlfriend Jennifer was asleep next to him, oblivious to Mullis's sudden inspiration. Mullis excitedly calculated that if this process was repeated 10 times, he could get over 1000 copies of any DNA fragment. 20 cycles would bring 1 million copies, and 30 cycles would be 1 billion copies [68, 69]! Mullis knew that his idea would change the rules of molecular biology. He returned to the lab and started the first experiment, trying to start with human DNA. Despite slow progress, Mullis did not give up. Finally, on the night of December 16, 1983, he successfully achieved rapid amplification of DNA. This achievement not only shocked Mullis but also shocked the entire scientific community. His invention was named Polymerase Chain Reaction (PCR) (Fig. 4.19).

However, the perfection of PCR technology could not be separated from another key discovery—Taq polymerase. In 1976, Chien Chia-yun, a female scientist from Taiwan, China, extracted Taq polymerase, which can withstand high temperatures, from Thermus aquaticus during her postgraduate studies in the Department of Biology at the University of Cincinnati. This enzyme was later used to replace the heat-sensitive E. coli DNA polymerase, greatly simplifying PCR work [4].

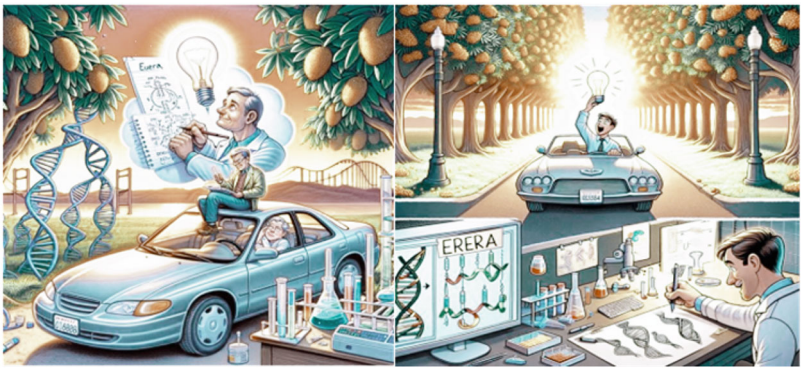


Fig. 4.19 The inspirational journey of Mullis's invention of PCR

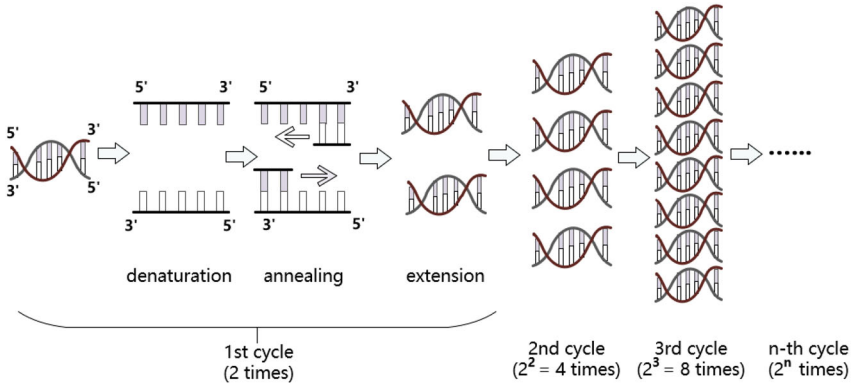


Fig. 4.20 Schematic diagram of the basic principle of PC

The invention of PCR technology completely changed the research methods of molecular biology, making rapid replication and analysis of DNA possible. This invention won Mullis the 1993 Nobel Prize in Chemistry, and his name was thus recorded in the annals of science.

4.4.2 Basic Principles

The basic principle of Polymerase Chain Reaction (PCR) is based on the replication mechanism of DNA molecules. In the PCR process, specific primers, DNA polymerase, and thermal cycling conditions work together on the DNA template to achieve rapid amplification of its specific sequence. This process involves three main steps: denaturation, annealing, and extension (Fig. 4.20) [4].

Denaturation Stage Under high temperature conditions (usually 94–98 °C), double-stranded DNA unwinds into two single strands. This step is achieved by breaking the hydrogen bonds in the DNA double helix structure.

Annealing Stage The temperature is lowered (usually 50–65 °C), allowing primers rich in specific sequences to bind specifically to the single-stranded DNA template. The design of the primers is crucial because they determine the specificity and efficiency of amplification.

Extension Stage At an appropriate temperature (usually 72 °C), DNA polymerase synthesizes new complementary DNA strands along the template strand. The choice of DNA polymerase is also very critical because it must maintain activity and stability under PCR cycling conditions.

Where, E represents the amplification efficiency, slope is the slope obtained from the logarithmic linear stage of the real-time PCR amplification curve [70]. The most famous PCR theory was proposed by Mullis in 1983, who found that by simulating the DNA replication mechanism in organisms, rapid amplification of DNA can be achieved under laboratory conditions [71]. The core of PCR technology lies in the choice of DNA polymerase and the design of primers, which directly affect the specificity and efficiency of the PCR reaction.

The amplification efficiency of PCR can be represented by the following formula:

$$E = (10^{-\frac{1}{slope}} - 1) \times 100\%$$

(1) Quantitative PCR (qPCR)

Quantitative PCR (qPCR), also known as real-time PCR, is an advanced technique for accurately quantifying the number of specific sequences in DNA or RNA samples. qPCR technology, by combining the amplification ability of traditional PCR and fluorescence detection technology, can monitor the quantity changes of target sequences in real time during the amplification process [72]. In qPCR, the DNA or reverse-transcribed RNA in the sample is first amplified by specific primers and DNA polymerase, and fluorescently labeled probes or dyes are used to detect the quantity of amplification products.

The key to qPCR is the real-time monitoring of the fluorescent signal, which is proportional to the initial amount of the target DNA sequence. During the amplification process, the increase in the fluorescent signal corresponds to each cycle of DNA replication, allowing for quantitative analysis of the amplification products. A fluorescent intensity signal is collected after each cycle, and the change in product quantity is monitored by the change in fluorescent intensity, finally resulting in a fluorescence amplification curve. The horizontal axis of the amplification curve represents the number of cycles, and the vertical axis represents the fluorescence intensity. Generally speaking, the fluorescence amplification curve can be divided into three stages: the fluorescence background signal stage (baseline period), the fluorescence signal exponential amplification stage, and the plateau

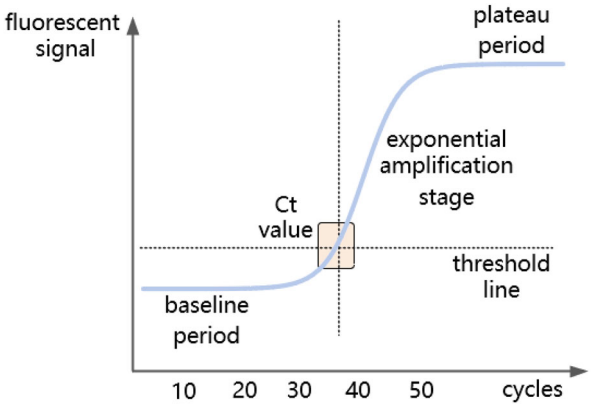


Fig. 4.21 qPCR amplification curve

period (Fig. 4.21). The choice of fluorescent probes is crucial because they directly affect the specificity and sensitivity of detection. Commonly used fluorescent probes include TaqMan probes and SYBR Green dyes [73].

QPCR technology has a wide range of applications in biological and medical research, especially in gene expression analysis, pathogen detection, and genetic disease diagnosis. For example, in infectious disease research, qPCR is used to quickly detect and quantify the DNA or RNA of pathogens, providing an efficient diagnostic tool [74]. In addition, qPCR plays an important role in cancer research, used to monitor the expression levels of cancer-related genes, which helps to understand the pathogenesis of cancer and develop new treatment strategies [75]. In bio-computation, this technique is mainly used to monitor the amplification of feasible solutions.

(2) Reverse Transcription PCR (RT-PCR)

Reverse Transcription PCR (RT-PCR) is a technique that combines reverse transcription and Polymerase Chain Reaction (PCR) to amplify specific DNA sequences from RNA samples. In RT-PCR, the RNA template is first transcribed into complementary DNA (cDNA) using reverse transcriptase, and then the cDNA is amplified using standard PCR technology [76]. This process makes RT-PCR a powerful tool for studying gene expression and viral load. Typically, RT-qPCR is divided into one-step and two-step methods (Fig. 4.22). One-step RT-qPCR combines reverse transcription with PCR amplification, allowing reverse transcriptase and DNA polymerase to complete the reaction in the same tube under the same buffer conditions. One-step RT-qPCR only requires the use of sequence-specific primers. In two-step RT-qPCR, the reverse transcription and PCR amplification processes are completed in two tubes, using different optimized buffers, reaction conditions, and primer design strategies.

During the reverse transcription stage, the choice of RNA template is crucial as it directly affects the quality of cDNA and the efficiency of subsequent amplification. The type of reverse transcriptase and reaction conditions also affect the efficiency

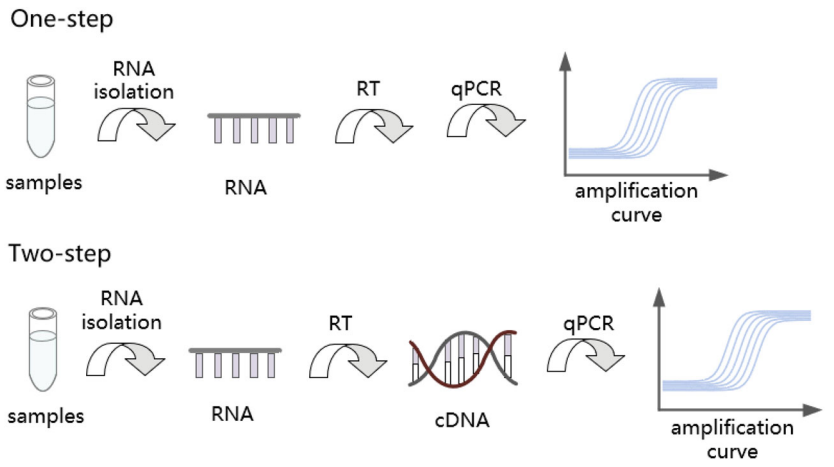


Fig. 4.22 Schematic diagram of "one-step" and "two-step" RT-PCR

and specificity of reverse transcription. Commonly used reverse transcriptases include M-MLV reverse transcriptase and AMV reverse transcriptase [77].

RT-PCR has a wide range of applications in biomedical research, especially in virology and cancer biology. For example, RT-PCR is used to quantify the RNA levels of HIV and hepatitis B virus, thereby assessing the activity of viral replication and treatment effects [78]. In cancer research, RT-PCR is used to detect and quantify the expression of tumor markers, which helps in early diagnosis and treatment monitoring of cancer [79].

(3) Digital PCR (dPCR)

Digital PCR (dPCR) is an advanced molecular biology technique used to precisely quantify the absolute number of specific sequences in DNA samples. Unlike traditional PCR and real-time quantitative PCR (qPCR), dPCR achieves detection and counting of individual molecules by dividing the DNA sample into thousands to tens of thousands of independent micro-reaction units [80]. In dPCR, each micro-reaction unit contains zero or more DNA template molecules, and PCR amplification is then carried out simultaneously in all units. The key to dPCR lies in sample partitioning and data analysis. Sample partitioning creates a large number of independent reaction units, each of which can be considered an independent PCR reaction. This partitioning allows dPCR to reduce sample variability and improve detection accuracy and repeatability. After amplification, the absolute number of target sequences can be directly determined by counting the positive and negative reaction units [81] (Fig. 4.23).

DPCR technology has a wide range of applications in biomedical research, especially in the detection of low abundance target sequences, single nucleotide variation (SNV) detection, and gene expression analysis in complex samples. For example, dPCR is used in cancer research for the detection of circulating

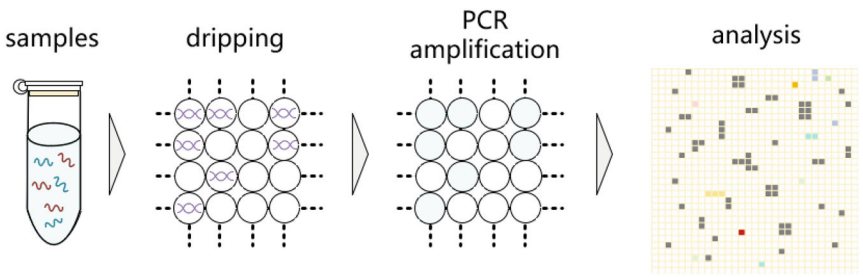


Fig. 4.23 Basic principle of digital PCR

tumor DNA (ctDNA), providing a non-invasive method for cancer diagnosis and monitoring [82]. In addition, dPCR has shown unique advantages in the diagnosis and treatment monitoring of genetic diseases [83]. In recent years, the development of dPCR technology has introduced new microfluidic chips and digital analysis methods, making dPCR a high-throughput, high-sensitivity analytical tool. These advances not only improve the convenience of dPCR operation, but also expand its application range in clinical diagnosis and biological research [84]. In bio-computation, this technology is mainly used to monitor the amplification and resolution of feasible solutions.

(4) Multiplex PCR (Multiplex PCR)

Multiplex PCR is an efficient molecular biology technique that allows multiple different DNA target sequences to be amplified simultaneously in a single reaction. This technique, by using multiple pairs of specific primers, combines the amplification capability of traditional PCR, achieving parallel detection and analysis of multiple gene sequences [85]. In multiplex PCR, each pair of primers specifically binds to different target sequences, and all target sequences are amplified in the same PCR reaction. The key to multiplex PCR technology lies in primer design and optimization of reaction conditions. Primers must have high specificity and minimal interaction to avoid non-specific amplification and primer dimerization. In addition, PCR reaction conditions, such as annealing temperature and cycle number, need to be precisely controlled to ensure effective amplification of all target sequences [86].

Multiplex PCR has a wide range of applications in clinical diagnosis, pathogen detection, genetic disease screening, and forensic science. For example, in infectious disease studies, multiplex PCR is used to simultaneously detect multiple pathogens, thereby improving the efficiency and accuracy of diagnosis [87]. In genetic research, multiplex PCR is used for rapid screening of multiple genetic markers, which helps in early diagnosis and risk assessment of genetic diseases [88]. In recent years, the development of multiplex PCR technology has introduced new detection platforms and analysis methods, such as real-time quantitative PCR and high-throughput sequencing, further improving its sensitivity and diversity. These advances not only improve the convenience of multiplex PCR operation, but also expand its application range in biomedical research and clinical applications [89].

With the advancement of next-generation bio-computation technologies, the concept of precise PCR has been proposed. This new concept technology overcomes the inherent disadvantages of the original PCR technology (such as replication errors) and is used for solution space resolution, making bio-computation a step closer to practicality and larger scale.

Appendix

Examples of commonly used restriction endonuclease recognition sequences and cutting points.

Name	**Source**	**Recognition Sequence**	**Cleavage Site**
EcoRI	Escherichia coli	5′GAATTC 3′CTTAAG	5′—G AATTC—3′ 3′—CTTAA G—5′
BamHI	Bacillus amyloliquefaciens	5′GGATCC 3′CCTAGG	5′—G GATCC—3′ 3′—CCTAG G—5′
HindIII	Haemophilus influenzae	5′AAGCTT 3′TTCGAA	5′—A AGCTT—3′ 3′—TTCGA A—5′
TaqI	Thermus aquaticus	5′TCGA 3′AGCT	5′—A CGA—3′ 3′—AGC A—5′
NotI	Nocardia otitidis	5′GCGGCCGC 3′CGCCGGCG	5′—GC GGCCGC—3′ 3′—CGCCGG CG—5′
HinfI	Haemophilus influenzae	5′GANTC 3′CTNAG	5′—G ANTC—3′ 3′—CTNA G—5′
Sau3A	Staphylococcus aureus	5′GATC 3′CTAG	5′— GATC—3′ 3′—CTAG —5′
PovII*	Proteus vulgaris	5′CAGCTG 3′GTCGAC	5′—CAG CTG—3′ 3′—GTC GAC—5′

Continued

SmaI*	Serratia marcescens	5′CCCGGG 3′GGGCCC	5′—CCC 3′—GGG	GGG—3′ CCC—5′
HaeIII*	Haemophilus egytius	5′GGCC 3′CCGG	5′—GG 3′—CC	CC—3′ GG—5′
AluI*	Arthrobacter luteus	5′AGCT 3′TCGA	5′—AG 3′—TC	CT—3′ GA—5′
EcoRV*	Escherichia coli	5′GATATC 3′CTATAG	5′—GAT 3′—CTA	ATC—3′ TAG—5′
Kpn [1]	Klebsiella pneumonia	5′GGTACC 3′CCATGG	5′—GGTAC 3′—C	C—3′ CATGG—5′
PstI [1]	Providencia stuartii	5′CTGCAG 3′GACGTC	5′—CTGCA 3′—G	G—3′ ACGTC—5′
SacI [1]	Streptomyces achromo-genes	5′GAGCTC 3′CTCGAG	5′—GAGCT 3′—C	C—3′ TCGAG—5′
SalI [1]	Streptomyces albue	5′GTCGAC 3′CAGCTG	5′—G 3′—CAGCT	TCGAC—3′ G—5′
SphI [1]	Streptomyces phaeochro-mogenes	5′GCATGC 3′CGTACG	5′—G 3′—CGTAC	CATGC—3′ G—5′
XbaI [1]	Xanthomonas badrii	5′TCTAGA 3′AGATCT	5′—T 3′—AGATC	CTAGA—3′ T—5′

*Produce blunt ends.

References

1. Harvey, F.L.: Molecular Cell Biology. Macmillan (2008).
2. Tetsuya, S., Akira, S., Petr, G., et al.: Error-prone bypass patch by a low-fidelity variant of dna polymerase zeta in human cells. DNA Repair **100**(103052) (2021).
3. Alice, C., David, B.E., John, M.T.: Deoxyribonucleic acid polymerase from the extreme thermophile thermus aquaticus. Journal of Bacteriology **127**(3), 1550–1557 (1976).
4. Randall, K.S., David, H.G., Susanne, S., et al.: Primer-directed enzymatic amplification of dna with a thermostable dna polymerase. Science **239**(4839), 487–491 (1988).
5. Randall, K.S., Stephen, S., Fred, F., et al.: Enzymatic amplification of Beta-globin genomic sequences and restriction site analysis for diagnosis of sickle cell anemia. Science **230**(4732), 1350–1354 (1985).
6. Frances, Lawyer, Susanne S., et al.: High-level expression, purification, and enzymatic characterization of full-length Thermus aquaticus DNA polymerase and a truncated form deficient in 5'to 3'exonuclease activity. Genome Research **2**(4), 275–287 (1993).
7. Kenneth, R.T., Thomas, A.K.: Fidelity of dna synthesis by the Thermus aquaticus DNA polymerase. Biochemistry **27**(16), 6008–6013 (1988).
8. Tamás, L., Huttová, J., Mistrk, I., et al.: Effect of carboxymethyl chitin-glucan on the activity of some hydrolytic enzymes in maize plants. Chemical Papers **56**(5), 326–329 (2002).
9. Allan, M.M., Walter, G.: [57] sequencing end-labeled DNA with base-specific chemical cleavages. Methods in Enzymology **65**(1), 499–560 (1980).
10. Ravinderjit, S.B., Nickolas, C., Cliff, J., et al.: Solution of a 20-variable 3-sat problem on a dna computer. Science **296**(5567), 499–502 (2002).
11. John, A.R.: The Fidelity of DNA Computation. The University of Memphis (1999).
12. Martyn A.: DNA computation. University of Warwick (1997).
13. Bin F.: Volume bounded molecular computation. Yale University (1997).
14. Tom, H., Grzegorz, R., Reno, S.B., et al.: Computing with DNA by operating on plasmids. Biosystems **57**(2), 87–93 (2000).
15. Stephen, P.A.F., Leighton, R., Michael, C.P., et al.: Light-directed, spatially addressable parallel chemical synthesis. Science **251**(4995), 767–773 (1991).
16. Editoral: To affinity . . . and beyond! Nature Genetics **14**(4), 367–370 (1996).
17. Jing, C., Edward, L.S., Lei, W., et al.: Preparation and hybridization analysis of DNA/RNA from E. coli on microfabricated bioelectronic chips. Nature Biotechnology **16**(6), 541–546 (1998).
18. Devashish, R., Lina, A., Archana, R., et al.: The CRISPR-Cas immune system: biology, mechanisms and applications. Biochimie **117**, 119–128 (2015).
19. Francisco, J.M.M., Díez-Villaseñor, García-Martínez: Short motif sequences determine the targets of the prokaryotic CRISPR defence system. Microbiology **155**(3), 733–740 (2009).
20. Elitza, D., Krzysztof, C., Cynthia, M.S., et al.: CRISPR RNA maturation by trans-encoded small RNA and host factor RNase III. Nature **471**(7340), 602–607 (2011).
21. Samuel, S.H., Redding, S., Jinek, M., et al.: DNA interrogation by the CRISPR RNA-guided endonuclease Cas9. Biophysical Journal **106**(2), 695a (2014).
22. Alexis, C.K., Yongjoo, B.K., Michael, S.P., et al.: Programmable editing of a target base in genomic DNA without double-stranded DNA cleavage. Nature **533**(7603), 420–424 (2016).
23. Nicole, M.G., Alexis, C.K., Holly, A.R., et al.: Programmable base editing of a• t to g• c in genomic DNA without DNA cleavage. Nature **551**(7681), 464–471 (2017).
24. John, P.G., David, B.T, David, R.L.: Fusion of catalytically inactive Cas9 to Fokl nuclease improves the specificity of genome modification. Nature Biotechnology **32**(6), 577–582 (2014).
25. Bernd, Z., Jonathan, S.G., Omar, O.A., et al.: Cpf1 is a single RNA-guided endonuclease of a class 2 CRISPR-Cas system. Cell **163**(3), 759–771 (2015).
26. Janice, S.C., Enbo, M., Lucas, B.H., et al.: Crispr-Cas12a target binding unleashes indiscriminate single-stranded dnase activity. Science **360**(6387), 436–439 (2018).

27. Shi-Yuan, L., Qiu-Xiang, C., Jia-Kun, L., et al.: CRISPR-Cas12a has both cis-and trans-cleavage activities on single-stranded DNA. Cell Research **28**(4), 491–493 (2018).
28. Xiaosa, L., Ying, W., Yajing, L., et al.: Base editing with a Cpf1-cytidine deaminase fusion. Nature Biotechnology **36**(4), 324–327 (2018).
29. Piro, S., John, Y., Timothy, K.L.: Synthetic circuits integrating logic and memory in living cells. Nature Biotechnology **31**(5), 448–452 (2013).
30. Lila, K.: Dna computing: arrival of biological mathematics. The Mathematical Intelligencer **19**(2), 9–22 (1997).
31. Tabatabaei, Y., Yongbo, Y., Jian, M., et al.: A rewritable, random-access DNA-based storage system. Scientific Reports **5**(1), 1–10 (2015).
32. Takafumi, M., Shiva, R., Robert, D., et al.: Synthesizing biomolecule-based Boolean logic gates. ACS Synthetic Biology **2**(2), 72–82 (2013).
33. Yuchen, L., Yayue, Z., Li, L., et al.: Synthesizing AND gate genetic circuits based on CRISPR-Cas9 for identification of bladder cancer cells. Nature Communications **5**(1), 5393 (2014).
34. Shaohua, G., Xi, W., Ping, Z., et al.: AND logic-gate-based CRISPR/Cas12a biosensing platform for the sensitive colorimetric detection of dual miRNAs. Analytical Chemistry **94**(45), 15839–15846 (2022).
35. Erik, W., Furong, L., Lisa, A.W., et al.: Design and self-assembly of two-dimensional DNA crystals. Nature **394**(6693), 539–544 (1998).
36. Xiaoping, Y., Lisa, A.W., Jing, Q., et al.: Ligation of DNA triangles containing double crossover molecules. Journal of the American Chemical Society **120**(38), 9779–9786 (1998).
37. Chuan, Z., Yu, H., Yi, C., et al.: Aligning one-dimensional DNA duplexes into two-dimensional crystals. Journal of the American Chemical Society **129**(46), 14134–14135 (2007).
38. Stefan, W.H., Steffen, J.S., Mark, B., et al.: The 2015 super-resolution microscopy roadmap. Journal of Physics D: Applied Physics **48**(44), 443001 (2015).
39. Bo, H., Mark, B., Xiaowei, Z.: Super-resolution fluorescence microscopy. Annual Review of Biochemistry **78**(1), 993–1016 (2009).
40. Eric, B., George, H.P., Rachid, S., et al.: Imaging intracellular fluorescent proteins at nanometer resolution. Science **313**(5793), 1642–1645 (2006).
41. Samuel, T.H., Thanu, P.K.G., Michael, D.M.: Ultra-high resolution imaging by fluorescence photoactivation localization microscopy. Biophysical Journal **91**(11), 4258–4272 (2006).
42. Michael, J.R., Mark, B., Xiaowei, Z.: Sub-diffraction-limit imaging by stochastic optical reconstruction microscopy (STORM). Nature Methods **3**(10), 793–796 (2006).
43. Mike, H., Sebastian, V.D.L., Mark, S., et al.: Subdiffraction-resolution fluorescence imaging with conventional fluorescent probes. Angewandte Chemie-International Edition **47**(33), (2008).
44. Alexey, S., Robin, M.H.: Wide-field subdiffraction imaging by accumulated binding of diffusing probes. Proceedings of the National Academy of Sciences **103**(50), 18911–18916 (2006).
45. Ralf, J., Maier, S.A., Johannes, B.W., et al.: Multiplexed 3D cellular super-resolution imaging with DNA-PAINT and Exchange-PAINT. Nature Methods **11**(3), 313–318 (2014).
46. Ralf, J., Maier, S.A., Mingjie, D., et al.: Quantitative super-resolution imaging with qPAINT. Nature Methods **13**(5), 439–442 (2016).
47. Hunter: Foundations of Colloid Science. Oxford University Press (1987).
48. Ferdinand, F.R.: Sur un nouvel effet de l'électricité galvanique. Mémoires de la Société Impériale des Naturalistes de Moscou **2**, 327–337 (1809).
49. Dorian, H., Marco, M., Paolo, V., et al.: Anodic aqueous electrophoretic deposition of titanium dioxide using carboxylic acids as dispersing agents. Journal of the European Ceramic Society **31**(6), 1041–1047 (2011).
50. Joseh, S., Russel: Molecular Cloning: A Laboratory Manual Cold Spring Harbor. Cold Spring Harbor Laboratory Press (2001).
51. Ling, Cooksley, Bates, et al.: Antibodies to the glutamate dehydrogenase of Plasmodium falciparum. Parasitology **92**(2), 313–324 (1986).
52. Baker, D.R.: Capillary Electrophoresis. New York: Wiley (1995).

53. James, W.J., Krynn, D.L.: Zone electrophoresis in open-tubular glass capillaries. Analytical Chemistry **53**(8), 1298–1302 (1981).
54. Norman, J.D., Jianzhong, Z.: How capillary electrophoresis sequenced the human genome. Angewandte Chemie International Edition **39**(24), 4463–4468 (2000).
55. Adam, T.W., Richard, A.M.: Ultra-high-speed dna sequencing using capillary electrophoresis chips. Analytical Chemistry **67**(20), 3676–3680 (1995).
56. Xiaoming, R., Amy, D.G., Carlowitz: Structural basis for il-1Alpha recognition by a modified dna aptamer that specifically inhibits il-1Alpha signaling. Nature Communications **8**(1), 810 (2017).
57. Chih-Ching, H., Zehui, C., Huan-Tsung, C., et al.: Protein- protein interaction studies based on molecular aptamers by affinity capillary electrophoresis. Analytical Chemistry **76**(23), 6973–6981 (2004).
58. Pohl, H.A.: Dielectrophoresis: The Behavior of Neutral Matter in Nonuniform Electric Fields. Cambridge University Press (1978).
59. Jones, T.B.: Electromechanics of Particles. Cambridge University Press (1995).
60. Hughes, M.P.: Nanoelectromechanics in Engineering and Biology. CRC Press (2002).
61. Jae-Woo, C., Allen, P., Demetri, P.: Optical detection of asymmetric bacteria utilizing electro orientation. Optics Express **14**(21), 9780–9785 (2006).
62. Sina, M., Fatima, H.L., Michael, P.H.: Effects of cell detachment methods on the dielectric properties of adherent and suspension cells. Electrophoresis **36**(13), 1493–1498 (2015).
63. Matthew, S.P., Yanting, Z., Nawarathna, K., et al.: Dielectrophoretic separation of platelets from diluted whole blood in microfluidic channels. Electrophoresis **29**(6), 1213–1218 (2008).
64. Herbert, A.P., Ira, H.: Separation of living and dead cells by dielectrophoresis. Science **152**(3722), 647–649 (1966).
65. Mariano, D., Trapani, N., Gianni, M.: Deparray^{TM} system: an automatic image-based sorter for isolation of pure circulating tumor cells. Cytometry Part A **93**(12), 1260–1266 (2018).
66. Albert A.: Biochemical and biological applications of isotachophoresis. Elsevier Scientific Publishing Company (1980).
67. Crevillén, A.G., Mercedes, F., Diez-Masa, J.C.: On-chip single column transient isotachophoresis with free zone electrophoresis for preconcentration and separation of Alpha-lactalbumin and Beta-lactoglobulin. Microchemical Journal **133**, 600–606 (2017).
68. Kary, B.M.: The unusual origin of the polymerase chain reaction. Scientific American **262**(4), 56–65 (1990).
69. Mullis: Dancing Naked in the Mind Field. Vintage (2000).
70. Rutledge, R.G., Cote, C.: Mathematics of quantitative kinetic PCR and the application of standard curves. Nucleic Acids Research **31**(16), e93-e93 (2003).
71. Mary, H.: DNA amplification and detection made simple (relatively). PLoS Biology **4**(7), e222 (2006).
72. Christian, A.H., Junko, S., Kenneth, J.L., et al.: Real time quantitative PCR. Genome Research **6**(10), 986–994 (1996).
73. Ursula, E.G., Christian, A.H., Mickey, W.: A novel method for real time quantitative RT-PCR. Genome Research **6**(10), 995–1001 (1996).
74. Mackay: Real-time PCR in the microbiology laboratory. Clinical Microbiology and Infection **10**(3), 190–212 (2004).
75. Bustin: Invited review quantification of mRNA using real-time reverse transcription PCR (RT-PCR), trends and problems. Journal of Molecular Endocrinology **29**, 23–39 (2002).
76. Stephen, A.B., Tania, N.: Pitfalls of quantitative real-time reverse-transcription polymerase chain reaction. Journal of Biomolecular Techniques: JBT **15**(3), 155 (2004).
77. Willard, M.F., Stephen, J.W., Kent, E.V.: Quantitative RT-PCR: pitfalls and potential. Biotechniques **26**(1), 112–125 (1999).
78. Drosten: Detection of mycobacterium tuberculosis by real-time PCR: A comparative study of is6110 and mpb64. Journal of Clinical Microbiology **41**(6), 3043–3047 (2003).
79. Dennis, J.S., William, G., Lovell, A.J., John, et al.: Studies of the HER-2/neu proto-oncogene in human breast and ovarian cancer. Science **244**(4905), 707–712 (1989).

80. Bert, V., Kenneth, W.K.: Digital PCR. Proceedings of the National Academy of Sciences **96**(16), 9236–9241 (1999).
81. Benjamin, J.H., Kevin, D.N., Donald, A.M., et al.: High-throughput droplet digital PCR system for absolute quantitation of DNA copy number. Analytical Chemistry **83**(22), 8604–8610 (2011).
82. Aaron, M.N., Scott, V.B., Jacqueline, T., et al.: An ultrasensitive method for quantitating circulating tumor dna with broad patient coverage. Nature Medicine **20**(5), 548–554 (2014).
83. Valérie, T., Deniz, P., Abdel, E.l.A., et al.: Detecting biomarkers with microdroplet technology. Trends in Molecular Medicine **18**(7), 405–416 (2012).
84. Elizabeth, D., Paul, H.D., Frank, M.: Digital PCR strategies in the development and analysis of molecular biomarkers for personalized medicine. Methods **59**(1), 101–107 (2013).
85. Henegariu, Heerema, Dlouhy, et al.: Multiplex PCR: critical parameters and step-by-step protocol. Biotechniques **23**(3), 504–511 (1997).
86. Mary, C.E., Richard, A.G.: Multiplex PCR: advantages, development, and applications. Genome Research **3**(4), S65-S75 (1994).
87. Elfath, M.E., Ahmed, M.A., Robert, J.C., et al.: Multiplex PCR: optimization and application in diagnostic virology. Clinical Microbiology Reviews **13**(4), 559–570 (2000).
88. C.M., Strom, I.M., Verma: Multiplex PCR for diagnosis of aids-related lymphomas. Methods in Molecular Medicine **70**, 77–87 (2002).
89. Schoske, Vallone, Ruitberg: High throughput multiplex PCR and amplicon quantitation for human identity testing. Analytical Biochemistry **316**(1), 1–9 (2003).

Chapter 5
DNA Coding Theory and Algorithms

DNA computing is an emerging computational model that has garnered significant attention due to its distinctive advantages at the molecular biological level. Since it was introduced by Adelman in 1994, this field has made remarkable progress in solving **NP**-complete problems, enhancing information security, encrypting images, controlling diseases, and advancing nanotechnology. A key challenge in DNA computing is the design of DNA coding, which aims to minimize nonspecific hybridization and enhance computational reliability. The DNA coding design is a classical combinatorial optimization problem focused on generating high-quality DNA sequences that meet specific constraints, including distance, thermodynamics, secondary structure, and sequence requirements. This chapter comprehensively examines the advancements in DNA coding design, highlighting mathematical models, counting theory, and commonly used DNA coding methods. These methods include the template method, multi-objective evolutionary methods, and implicit enumeration techniques.[1]

5.1 Introduction

In the age of big data, social development confronts unprecedented challenges, one of the most significant being electronic computers' limitations in addressing **NP**-complete problems. As the scale of these problems grows, the computational effort required expands exponentially, often exceeding the capabilities of traditional computing models. This situation has created an urgent need to explore new computational tools to overcome the "bottleneck" issues that impede societal advancement. DNA computing, an emerging paradigm in computation, has demonstrated remarkable potential, particularly in solving **NP**-complete problems, due

[1] The content of this chapter is derived from the research paper [1].

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_5

to its high degree of parallelism. Since Adleman [2] first introduced the concept of using DNA molecules to tackle the 7-vertex Hamiltonian path problem in 1994, DNA computing has evolved into a groundbreaking approach that utilizes DNA molecules as information carriers, employing biochemical operations such as enzymes and PCR as computational mechanisms. Since then, various DNA computing models have been proposed and have successfully addressed numerous **NP**-complete problems [3], including the SAT problem [4, 5], the traveling salesman problem [6], and graph coloring problems [7]. Moreover, DNA computing models have been applied to numerous other areas, such as data encryption [8] and logic circuit constructions [9, 10].

Although DNA computing holds significant promise, its practical applications still encounter several challenges. The biochemical processes on which DNA computing relies are susceptible to errors. For example, PCR amplification has an efficiency of around 90%, while enzyme efficiency ranges from 80% to 95%. These errors can accumulate as the computational process iterates, affecting the results' accuracy. Furthermore, the fundamental operation in DNA computing, hybridization, can occur even when the DNA strands are not entirely complementary. This non-specific hybridization can lead to undesirable secondary structures, which deviate from the original design and potentially lead to incorrect computational results. These challenges constrain the scale of problems that DNA computing can tackle effectively. Currently, the largest successfully addressed problem by DNA computing involves identifying a 3-coloring in a graph consisting of 61 vertices [7].

To effectively scale DNA computing for larger problems, designing high-quality DNA codes presents a significant challenge [11]. DNA computing transforms the problem to be solved into DNA sequences, with solutions represented by DNA molecules formed through specific hybridization. However, low-quality DNA sequences can result in non-specific hybridization, inconsistent melting temperatures, and even computational failures. Consequently, developing reliable DNA sequences is essential for enhancing the efficiency of DNA computing. This not only influences the accuracy and efficiency of computations but also critically affects the scalability and reliability of DNA computing in practical applications. When designing DNA codes, we must consider multiple factors, such as melting temperatures (Tm), pairing specificity, stability, computational efficiency, and resource consumption. Research in this field is vital for advancing DNA computing and serves as a foundation for realizing its potential in tackling complex problems.

The DNA coding problem involves identifying large sets of single-stranded DNA molecules that meet specific requirements. When designing DNA sequences of length n, the solution space expands to 4^n, making the task of finding appropriate DNA sequences particularly challenging. This problem is inherently a combinatorial optimization problem and has been classified as **NP**-complete [12]. In the design process, considering too few constraints may compromise the quality of the designed DNA codes. Conversely, applying excessive constraints can significantly restrict the feasible solution space and increase computational complexity, making the problem even more difficult to solve. Moreover, optimizing DNA codes often entails navigating multiple conflicting constraints, such as distance requirements

and melting temperature. As a result, various optimization objectives must be considered during the design process, adding further complexity to the task. Achieving a balance among these constraints while identifying high-quality DNA codes remains a considerable challenge.

5.1.1 An Overview of the Advancement in DNA Coding Design

Various optimization algorithms have been developed over the years to address the challenges associated with DNA code design. In 1996, Deaton et al. [13] were among the first to apply genetic algorithms (GA) to generate DNA sequences, exploring the effects of temperature on DNA hybridization. Although this method effectively produced valid DNA codes under fewer constraints, its performance notably declined as the constraints increased [14, 15]. In 1997, Frutos et al. [16] proposed a template-based coding method to minimize sequence similarity by factoring in both Hamming and reverse-complement Hamming distances. This methodology generated a set of DNA sequences through two binary coding sets that met various constraints. Subsequent research further refined and extended its properties and optimization techniques [17–19].

Search algorithms, such as exhaustive search and implicit enumeration, are vital for DNA coding design. These algorithms systematically explore the potential solution space to identify optimal or near-optimal DNA sequences. While exhaustive search ensures the generation of constraint-satisfying sequences, its computational time is often lengthy, making it suitable mainly for small-scale problems [20]. Implicit enumeration has emerged as a widely used technique to overcome this limitation, employing progressive edge extension searches to enhance the coding process [21]. Given the vastness of the solution space, the random search method has been introduced as an effective alternative to traditional exhaustive search [22]. This method generates random DNA sequences and evaluates them against predefined criteria. The random search algorithm can discover high-quality solutions by exploring the solution space unbiasedly. However, it may struggle to converge to the optimal solution efficiently. As a result, random search is often combined with other methods, such as heuristic algorithms [23, 24]. A variety of heuristic algorithms have been developed for the design of DNA codes. Notable methods include simulated annealing [25], particle swarm optimization [26], and hybrid optimization that combines genetic algorithms with simulated annealing [27]. Other noteworthy approaches involve multi-objective chaotic evolutionary optimization [28], local search techniques [29], and colony optimization [30]. Furthermore, DNA coding research has utilized mathematical modeling and graph theory. For example, Caserta et al. [31] utilized mathematical programming to create a heuristic algorithm for generating DNA sequences. Similarly, Fouilhoux et al. [32] applied bipartite graph partitioning to address challenges in DNA coding. Marco et al. [33] combined mathematical programming with metaheuristic approaches to optimize the design of DNA sequences.

Researchers have focused on enhancing and refining constraint conditions to improve the quality of DNA codes. Garzon et al. [34] proposed the H-measure constraint, which calculates the minimum Hamming distance between one sequence and another by considering all possible shifts. Feldkamp et al. [35] introduced a method for defining similarity that restricts the maximum length of common subsequences between DNA sequences and includes a parameter to quantify inter-sequence similarity. This approach limits identical or complementary subsequences to a maximum length k; however, determining an optimal value for k remains challenging. To address diverse application needs, more targeted constraints have been developed, such as reverse-complement Hamming distance [26–28, 36], self-complementary Hamming distance [21, 25], and 3'-end H-measure constraints [25]. Additionally, secondary structure constraints and continuity constraints are crucial for coding design [37, 38]. Penchovsky et al. [22] explored the hybridization capabilities of DNA strands within hairpins, inner loops, and bulges, employing thermodynamic data to assess duplex stability. They used dynamic programming to create highly specific hybridizing DNA strands. Tanaka et al. [39] introduced a greedy filtering algorithm to enhance the efficiency of free energy calculations in thermodynamic-based methods, thus reducing design time. Together, these studies propel DNA coding design toward greater efficiency, reliability, and precision.

Although significant advancements in DNA sequence design, various methods often address distinct sets of constraints, many of which can conflict with one another. Consequently, identifying DNA sequences that excel across multiple constraints remains a considerable challenge. Multi-objective optimization algorithms present a robust approach to overcoming this issue. For instance, Shin et al. [40] developed the NACST/Seq system, outperforming traditional genetic algorithms and simulated annealing using constrained multi-objective optimization. Integrating multi-objective optimization into DNA coding design has notably improved the reliability of DNA sequences. In 2014, Chaves-González et al. [12] introduced MO-FA, a firefly algorithm specifically tailored for multi-objective optimization, capable of generating dependable sequences. In 2019, Chaves-González et al. [41] expanded their efforts by formulating the DNA coding challenge as a constrained multi-objective optimization problem consisting of four objectives and two constraints and proposed the pMO-ABC algorithm based on multi-objective evolutionary algorithms (MOEA), which yielded highly robust sequences. Recently, Xie et al. [42] introduced an enhanced billiard arithmetic optimization algorithm (BHAOA) for designing DNA codes, achieving sequences of high reliability. Similarly, Yang et al. [43] proposed continuous base-pairing constraints and developed the H-ACO algorithm, which demonstrated impressive performance in DNA sequence design, validated through NUPACK. Despite these progressions, conflicts among multiple constraints continue to pose significant challenges in DNA coding design. Future research is expected to focus on balancing these competing objectives to improve algorithm efficiency and code reliability further.

5.1.2 Organization

The remainder of this chapter is structured as follows. Section 5.2 discusses the key factors that influence the design of DNA codes and establishes mathematical models for the DNA coding problem, followed by a categorization of existing DNA coding algorithms. Section 5.3 investigates the theory of coding counts under the specified GC content condition, providing a systematic analysis of its theoretical foundations. Subsequent sections introduce several common DNA coding methods: Sect. 5.4 discusses template-based DNA coding methods, Sect. 5.5 focuses on multi-objective optimization coding methods, and Sect. 5.6 introduces implicit enumeration algorithms.

5.2 DNA Coding Problem

In DNA computing, essential processes such as solution generation, extraction, PCR amplification, and detection depend on DNA molecule hybridization. This specific hybridization not only forms the foundation of DNA computing but also plays a crucial role in determining the efficiency and accuracy of the computational process. However, the inherent characteristics of DNA molecules may result in hybridization errors, significantly affecting DNA computing processes' reliability and accuracy. Hybridization errors can be classified into false positives and false negatives [44, 45]. A false positive occurs when incompletely complementary DNA molecules erroneously hybridize, forming a double-stranded molecule. Figure 5.1 illustrates various DNA hybridization patterns. False positives generally arise from a sufficient degree of reverse complementarity or similarity among DNA molecules. In contrast, a false negative refers to the unsuccessful hybridization of completely complementary DNA molecules, which prevents the formation of a double strand with the intended target molecule. False negatives typically result from improper reaction conditions or errors in biochemical procedures. A robust coding strategy is vital to effectively reducing the incidence of false positives and false negatives. Rational DNA sequences can ensure that the computational process (biochemical reaction) closely adheres to a predefined model, thereby enhancing accuracy and efficiency in DNA computing. Several factors influence the realization of specific hybridization, including the distance between DNA sequences and various

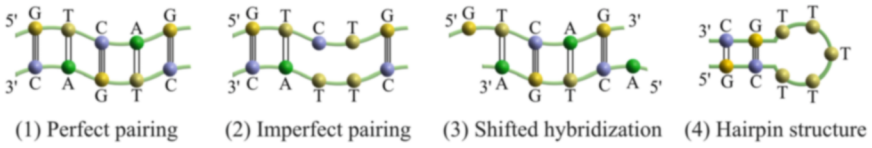


Fig. 5.1 Four types of DNA hybridization patterns

biochemical constraints, such as similar thermodynamic properties, the avoidance of secondary structure formation, and the presence of specific enzyme cleavage sites. Collectively, these factors contribute to the reliability and efficiency of the biochemical reaction system. A more detailed examination of these constraints is provided below.

In the subsequent discussion, let $S(n)$ denote the set of all single-stranded DNA sequences of length n, where n is a positive integer. For any sequence $x \in S(n)$, we define x^r and x^c to represent the reverse and complement sequences of x, respectively. Specifically, if the sequence is expressed as $x = 5'\text{-}x_1x_2\ldots x_n\text{-}3'$ (with $x_i \in \{\text{A, G, C, T}\}$ for $i = 1, 2, \ldots, n$), then its reverse sequence x^r is given by $x^r = 3'\text{-}x_nx_{n-1}\ldots x_1\text{-}5'$, and its complementary sequence x^c is defined as $x^c = 3'\text{-}x_1^cx_2^c\cdots x_n^c\text{-}5'$, where each base x_i and its complementary sequence x_i^c (for $i = 1, 2, \ldots, n$) conform to the Watson-Crick (WC) base pairing rules, which are as follows: if $x_i = \text{A}$, then $x_i^c = \text{T}$; if $x_i = \text{T}$, then $x_i^c = \text{A}$; if $x_i = \text{C}$, then $x_i^c = \text{G}$; and if $x_i = \text{G}$, then $x_i^c = \text{C}$. Additionally, we define $x^{rc} = (x^r)^c$ and $x^{cr} = (x^c)^r$. Notably, it holds that $x^{rc} = x^{cr}$x. We refer to x^{rc} as the reverse-complementary sequence of x.

5.2.1 Constraints in DNA Coding Design

DNA coding design involves several important constraints that must be carefully considered to achieve successful results. The primary factors include distance constraints, thermodynamic constraints, secondary structure constraints, and sequence constraints. Each of these elements is crucial for ensuring the reliability and functionality of the designed DNA sequences.

5.2.1.1 Distance Constraints

DNA sequence specificity is essential for accurate information representation and effective computation. Distance constraints are the principal means of maintaining the integrity of the sequence specificity of DNA interactions. The differences between DNA sequences are primarily evaluated using Hamming distance. Coding constraints based on Hamming distance can prevent nonspecific hybridization between DNA sequences, ensuring that each DNA strand pairs exclusively with its complementary counterpart to form a stable double helix structure. The most commonly employed distance constraints include the following seven types. Let $x = 5'\text{-}x_1x_2\cdots x_n\text{-}3'$ and $y = 5'\text{-}y_1y_2\cdots y_n\text{-}3'$ denote two single-strand DNA sequences of length n, where $x_i, y_i \in \{\text{A, G, C, T}\}$ represent the respective nucleotide bases.

(1) Hamming Distance [14] The Hamming distance between two DNA sequences, x and y, is the count of positions where the corresponding bases differ. In DNA

coding design, this distance is a crucial measure of dissimilarity between sequences. A larger Hamming distance between x and y signifies a greater number of differing positions, thereby decreasing the likelihood of specific hybridization between x and $y's$ complementary sequence y_c, as well as between y and $x's$ complementary sequence x_c. The Hamming distance between the sequences x and y is formally defined as follows:

$$H(x, y) = \sum_{i=1}^{n} h(x_i, y_i), \tag{5.1}$$

where $h(x_i, y_i) = 0$ if $x_i = y_i$ and $h(x_i, y_i) = 1$ if $x_i \neq y_i$.

(2) H-measure [34] The H-measure addresses the potential for shift hybridizations between DNA molecules by restricting the degree of complementary base pairing between two DNA sequences. The H-measure of two DNA sequences x and y, denoted by $H\text{-}measure(x, y)$, is defined as the smallest shifted Hamming distance between x and the reverse complementary sequence y^{rc} of y. The calculation of the H-measure is expressed using the following formula:

$$H\text{-}measure(x, y) = \min_{-n<k<n} H(x, \sigma^k(y^{rc})) \tag{5.2}$$

where the expression $\sigma^k(y^{rc})$ refers to the DNA sequence obtained by shifting y^{rc} to the right by k positions. The H-measure offers a more precise evaluation of dissimilarity between two sequences. A smaller value of $H\text{-}measure(x, y)$ indicates a greater similarity between x and y^{rc}, signifying an increased likelihood of shifting hybridizations between x and y. In contrast, a larger value suggests a decreased probability of such hybridizations occurring. Given the complexity of DNA hybridization, the shifting distance provides a straightforward computational metric for assessing differences between sequences.

In optimization-driven coding methodologies, the H-measure evaluation function is framed as a minimization problem [25]. Given a set S consisting of m DNA sequences, each of length n, the H-measure for an individual sequence $x \in S$ is defined in a specific manner.

$$H\text{-}measure\,(x) = \max_{y \in S, y \neq x} \max_{-n<k<n} \left(n - H(x, \sigma^k(y^{rc}))\right) \tag{5.3}$$

To further enhance the accuracy of hybridization assessments between DNA sequences, Shin et al. [40] introduced a gapped version of the H-measure evaluation function (Eq. 5.3). Specifically, for a sequence x in S, this function is defined as follows:

$$H\text{-}measure\,(x) = \sum_{y \in S,\ y \neq x} \max_{0 \leq g \leq n} \max_{0 \leq k \leq n+g-1} C(x^{(\_)g}x, \sigma^k(y^r)) \tag{5.4}$$

In Eq. 5.4, $(\_)g$ represents the insertion of g gaps into the sequence x. The function $C(x^{(\_)g}x, \sigma^k(y^r))$ determines the number of complementary base pairs between $x^{(\_)g}x$ and $\sigma^k(y^r)$. Furthermore, incorporating penalties for continuous complementary regions can significantly improve the effectiveness of the H-measure constraint [12]. A higher $H\text{-}measure(x)$ value indicates an increased likelihood of hybridization between the sequence x and sequences in $S\backslash\{x\}$, whereas a lower value suggests a reduced likelihood of hybridization.

(3) Similarity [16] The similarity constraint is utilized to evaluate the degree of similarity of a sequence with other sequences based on their base composition. DNA sequences satisfying this constraint are designed to ensure that their subsequences remain as unique as possible in the same direction, while also avoiding any repetition during shifting conditions. For a set S consisting of m DNA sequences, each of length n, the similarity evaluation function for a DNA sequence $x \in S$ is defined as follows:

$$\text{Similarity}(x) = \max_{y \in S,\ y \neq x} \max_{-n<k<n} \left(n - H(x, \sigma^k(y))\right) \tag{5.5}$$

Again, Shin et al. [40] introduced a gapped similarity evaluation function, which is defined as follows:

$$\text{Similarity}(x) = \sum_{y \in S, y \neq x} \max_{0 \leq g \leq n} \max_{0 \leq k \leq n+g-1} E(x^{(\_)g}x, \sigma^k(y)) \tag{5.6}$$

where the function $E(x^{(\_)g}x, \sigma^k(y))$ quantifies the number of identical bases between $x^{(\_)g}x$ and $\sigma^k(y)$. Additionally, the effectiveness of the similarity constraint can be significantly improved by introducing penalties for consecutive identical bases [12].

(4) Reverse-Complementary Hamming Distance [17] Sequence x has the potential to hybridize with the reverse sequence y. The reverse-complementary Hamming distance, represented as $H^{rc}(x, y)$, quantifies the degree of similarity between x and the reverse-complementary sequence of y, which is defined as

$$H^{rc}(x, y) = H(x, y^{rc}) \tag{5.7}$$

(5) Self-complementary Hamming Distance [25] To prevent a DNA molecule from hybridizing with its reverse sequence, the concept of self-complementary Hamming distance is introduced, represented as $H_s(x)$. This concept is defined as the minimum shifted Hamming distance between the sequence x and its reverse-complementary sequence, that is,

$$H^s(x) = \min_{-n<k<n} H(x, \sigma^k(x^{rc})) \tag{5.8}$$

(6) Completely Complementary at 3'-end [25] If the 3' end of a DNA sequence is complementary to a portion of another sequence, erroneous amplification may occur during PCR. This issue can be avoided using the constraint of completely complementary at 3'-end. This evaluation function between sequences x and y is denoted as H-measure_end(x, y), and is defined as follows:

$$\text{H-measure\_end}(x, y) = CN(x, y^{(k)}) \tag{5.9}$$

where $CN(x, y^{(k)})$ denotes the number of completely complementary sites between sequence x and the k-base sequence derived from the 3' end of sequence y. The parameter k is predetermined.

(7) Overlapping Subsequences [46] This constraint requires that no two subsequences of length m may be identical, thereby preventing the occurrence of long, continuous identical subsequences within DNA sequences. As a result, this measure significantly reduces the overall similarity between the sequences.

In addition to the previously discussed constraints, there are some straightforward extensions of the Hamming distance. For example, the reverse Hamming distance, denoted as $H^r(x, y)$, is defined as the Hamming distance between sequence x and the reverse of sequence y, i.e., $H^r(x, y) = H(x, y^r)$. Likewise, the complement Hamming distance, represented as $H^c(x, y)$, is defined as the Hamming distance between sequence x and the complement of sequence y, i.e., $H^c(x, y) = H(x, y^c)$.

5.2.1.2 Thermodynamic Constraints

In contrast to traditional coding methods that depend on Hamming distance constraints, thermodynamic constraints provide a more accurate evaluation of the stability and specificity of DNA sequences. This approach involves quantifying both the melting temperature and the free energy changes of DNA strands, effectively preventing non-specific hybridization. By integrating thermodynamic constraints, we can comprehensively analyze the interactions between sequences and their behavior under specific environmental conditions, offering enhanced scientific and practical guidance for DNA sequence design.

(1) Melting Temperature [25] A vital biochemical process in DNA computing is the denaturation of double-stranded DNA molecules. Due to the large number of DNA molecules involved in biochemical reactions and the necessity of denaturing all target DNA molecules within a brief period, it is essential for the DNA molecules designated as "data" to have similar melting temperatures. The melting temperature (T_m) of a single-stranded DNA sequence x is defined as the temperature at which 50% of the base pairs in the DNA molecule formed by x and its complementary strand become denatured. This parameter plays a critical role in evaluating the thermodynamic stability of DNA molecules.

The (T_m) value of a DNA molecule is determined by various factors, including its concentration, solution pH value, molecular size, GC content, and the arrangement of its base sequence. Theoretically, the melting temperature can be calculated using a formula that is grounded in thermodynamic principles [47].

$$T_m = \frac{\Delta H^\circ}{\Delta S^\circ + R \ln C_t} \tag{5.10}$$

where ΔH° and ΔS° denote the changes in enthalpy and entropy that occur during the hybridization reaction. The gas constant R is valued at 1.987 cal/K·mol. Additionally, C_t represents the molar concentration of the DNA molecule, with symmetric sequences expressed as $C_t/4$. Generally, a greater degree of hybridization pairing between DNA sequences corresponds to a more stable double-helix structure and, as a result, a higher melting temperature.

For a DNA sequence of length n, denoted as $x_1 x_2 \ldots x_n$, the free energy and entropy changes ΔH° and ΔS° can be approximated using the following formula:

$$\Delta X = \theta + \sum_{i=1}^{n-1} w(x_i, x_{i+1}) \tag{5.11}$$

where θ is a correction factor, and $w(x_i, x_{i+1})$ represents the negative enthalpy or entropy weight for a 2-bp base pair $x_i x_{i+1}$ (as listed in Table 5.1).

(2) Free Energy [19] The free energy change (ΔG) reflects the energy variation that occurs when two single-stranded DNA molecules hybridize to form a double-stranded DNA structure. Since DNA hybridization generally releases heat, ΔG is typically negative ($\Delta G < 0$). The magnitude of ΔG serves as an essential

Table 5.1 Nearest-neighbor thermodynamic parameters for Watson-Crick base pairs [48]

Base pair sequence 5' →3'/3' →5'	ΔH° (kcal/mol)	ΔS° (e.u.)	ΔG° (kcal/mol)
AA/TT	−7.6	−21.3	−1.00
AT/TA	−7.2	−20.4	−0.88
TA/AT	−7.2	−21.3	−0.58
CA/GT	−8.5	−22.7	−1.45
GT/CA	−8.4	−22.4	−1.44
CT/GA	−7.8	−21.0	−1.28
GA/CT	−8.2	−22.2	−1.30
CG/GC	−10.6	−27.2	−2.17
GC/CG	−9.8	−24.4	−2.24
GG/CC	−8.0	−19.9	−1.84
Initiation	+0.2	−5.7	+1.96
Terminal AT Penalty	+2.2	+6.9	+0.05
Symmetry Correction	0.0	−1.4	+0.43

indicator for assessing the stability of the double-stranded DNA; the greater the absolute value of ΔG, the more stable the resulting structure is. To prevent non-specific hybridization between DNA sequences, a minimum free energy constraint is imposed on the DNA set C. Specifically, a threshold for the minimum free energy change, $\Delta G_{\min}$, is established, requiring that the ΔG for non-specific hybridization between any two DNA molecules in C exceeds $\Delta G_{\min}$. This approach ensures that stable double-strand structures cannot form, thereby effectively avoiding non-specific hybridization.

The free energy is calculated using the Nearest-Neighbor thermodynamic model [47], and the formula is as follows:

$$\Delta G = \sum_i n_i \Delta G(i) + \Delta G_{\text{ini GC}} + \Delta G_{\text{ini AT}} + \Delta G_{\text{sym}} \tag{5.12}$$

where $\Delta G(i)$ represents the free energy associated with a neighboring base pair. For instance, $\Delta G(1)$ corresponds to ΔG(AA/TT), $\Delta G(2)$ corresponds to ΔG(TA/AT), and this continues for a total of 10 possible Watson-Crick nearest-neighbor base pair combinations. The variable n_i denotes the frequency of each $\Delta G(i)$. Furthermore, $\Delta G_{\text{ini GC}}$ and $\Delta G_{\text{ini AT}}$ are correction terms used for starting positions containing GC or AT pairs, respectively. Lastly, ΔG_{sym} serves as the correction term for self-complementary DNA sequences.

5.2.1.3 Secondary Structure Constraints [27]

Single-stranded DNA molecules have the potential to spontaneously fold into intricate secondary structures, such as hairpin loops, because of the presence of reverse complementary subsequences. These secondary structures can disrupt DNA computation during synthesis, sequencing, and solution detection. Consequently, DNA coding design generally strives to prevent the formation of secondary structures during both the coding phase and solution generation. For a DNA sequence x with a length of ℓ, the computation of its hairpin structure is defined as follows:

$$\text{Hairpin}(x) = \sum_{p=P_{\min}}^{\lfloor(\ell-R_{\min})/2\rfloor} \sum_{r=R_{\min}}^{\ell-2p} \sum_{i=1}^{\ell-2p-r} T\left(\sum_{j=1}^{p} bp(x_{p+i-j}, x_{p+i+r+j}), \frac{p}{2}\right) \tag{5.13}$$

In Eq. 5.13, p denotes the stem length of the hairpin, which is defined as

$$p = \text{pinlen}(p, r, i) = \min(p + i, \ell - r - i - p),$$

where $P_{\min}$ is the minimum stem length, r is the loop length of the hairpin, and $R_{\min}$ is the minimum loop length. The function $T(a, T_{\text{value}})$ is a threshold function

that returns a if a exceeds T_{value}; otherwise, it produces a result of 0. In addition, the function $bp(b, b')$ evaluates whether the bases b and b' are complementary. If the bases are found to be complementary, the function returns 1; if they are not, it returns 0.

5.2.1.4 Sequence Constraints

In DNA computing, sequence constraints are crucial for ensuring sequence specificity and the reliability of experiments. These constraints can be categorized into several types:

(1) GC Content [49] In double-stranded DNA, adenine (A) pairs with thymine (T) via two hydrogen bonds, while guanine (G) pairs with cytosine (C) through three hydrogen bonds. Consequently, GC content is a critical factor influencing the melting temperature (T_m) and free energy change (ΔG). By adjusting the GC content, it is possible to keep the melting temperature and free energy change within a narrow range. Typically, a GC content of approximately 50% is recommended for the design of PCR primers and for DNA computing coding design.

(2) Continuity [25] The consecutive occurrence of identical nucleotide bases within a DNA sequence can lead to the formation of undesirable secondary structures, primarily due to hydrogen bonding interactions among the bases. Therefore, to maintain the stability of the sequence and preserve the integrity of experimental results, it is a standard practice in sequence design to restrict the length of consecutive identical bases. Specifically, for a sequence x of length ℓ, the calculation for its continuity is defined as follows:

$$\text{Continuity}(x) = \sum_{i=1}^{\ell-t+1} \sum_{\alpha \in \{\text{A, T, G, C}\}} \left(T\left(C_\alpha(x, i), t\right)\right)^2 \tag{5.14}$$

where the function $C_\alpha(x, i)$ returns a value c if there exists an integer c such that $x_i \neq \alpha$, $x_{i+j} = \alpha$ (for $1 \leq j \leq c$), and $x_{i+c+1} \neq \alpha$. Otherwise, it returns 0. Here, x_i represents the i-th base of the sequence x.

(3) Base [50] Base constraints involve deliberately limiting the use of specific DNA bases within a sequence to fulfill experimental requirements or to optimize performance. For example, certain experiments may need specific bases to be retained for designated functions, while others might restrict certain bases to minimize potential interference. Research has shown that utilizing only A, T, and C while excluding G can significantly decrease hybridization and the stability of secondary structures [4, 51, 52]. Though this approach leads to reduced sequence diversity, it ultimately enhances the specificity of DNA libraries and the experiments' reliability.

(4) Special Subsequence [16] Enzymes are biological molecules with specific functions, primarily proteins, that play crucial roles in DNA manipulation. For

example, ligases connect DNA strands, while restriction endonucleases can cleave DNA at specific locations. These enzymes greatly enhance the tools available for DNA computing, allowing for greater flexibility in algorithm design and improving both the computational processes and solution detection. It's important to recognize that these enzymes are programmed to identify specific base sequences. Consequently, when designing DNA sequences, avoiding these particular functional subsequences is essential to prevent errors in biological experiments. Additionally, certain sequences, such as codons, must be excluded to avoid inadvertently triggering unwanted molecular reactions under specific experimental conditions. By carefully excluding these sequences, researchers can minimize experimental risks and significantly enhance the reliability and success rate.

5.2.2 DNA Coding Problem and Its Mathematical Model

The DNA coding problem encompasses three primary factors: coding quantity, length, and quality. Various constraints influence the quality of coding, including distance, thermodynamics, secondary structure, and sequence constraints. The stricter these constraints are, the higher the quality of the coding, resulting in greater reliability and stability in the biochemical reactions during the computation process. Under the specified constraints, longer coding lengths lead to more coding possibilities; however, this also increases synthesis costs and complicates the control of long DNA chains in experimental settings. Conversely, shorter coding lengths provide fewer coding possibilities, lower synthesis costs, and simpler experimental control. Let $x = 5'\text{-}x_1x_2\ldots x_n\text{-}3'$ represent a single-stranded DNA sequence of length n, where $x_i \in \{\text{A, G, C, T}\}$ signifies the nucleotide bases. Define $S(n)$ as the set of all single-stranded DNA sequences of length n; therefore, the size of $S(n)$ is represented by $|S(n)| = 4^n$. The aim of DNA coding design is to identify a subset $S \subseteq S(n)$ such that the DNA sequences in S conform to various specified constraints.

Given the interdependence among coding quantity, length, and quality, the DNA coding problem is characterized as a multi-constrained optimization challenge. Below, two equivalent definitions of the DNA coding problem are presented: the *minimum coding length problem* and the *maximum coding set problem*.

Definition 5.1 (Minimum Coding Length Problem) Given a set of constraint criteria $C = \{f_1, f_2, \ldots, f_\ell\}$ and a specified coding quantity N, the objective of the minimum coding length problem is to identify the smallest positive integer n such that the set $S(n)$ contains a subset $S \subseteq S(n)$ that meets the following requirements: $|S| \geq N$; and for every DNA sequence s in S, each condition $f_i(s)$ must hold for all constraints C_i where $i = 1, 2, \ldots, \ell$.

Definition 5.2 (Maximum Coding Set Problem) Given a set of constraint criteria $C = \{f_1, f_2, \ldots, f_\ell\}$ and a coding length n, the goal of the maximum coding set

problem is to identify the largest subset $S \subseteq S(n)$ such that for every DNA sequence s in S, the condition $f_i(s)$ is satisfied for each constraint C_i where $i = 1, 2, \ldots, \ell$.

These two definitions are fundamentally equivalent. Definition 5.1 is concerned with identifying the smallest sequence length n such that the solution space of 4^n includes at least N sequences that comply with the given constraints. In contrast, Definition 5.2 focuses on selecting the maximum subset of sequences of length n that meet these constraints within the 4^n solution space. In practical applications, constraints are typically developed based on specific needs, and due to the limitations of biochemical operations, coding lengths are generally restricted to fewer than 50 bases. Consequently, algorithm design often prioritizes Definition 5.2, which aims to identify the largest possible coding set for a specified coding length. Below, we provide an example of the maximum coding set problem based on Definition 5.2, utilizing the constraint criteria outlined in Sect. 5.2.1.

The set of constraint criteria $C = \{f_1, f_2, f_3, f_4\}$ is defined as follows:

- Distance constraint (f_1): For any two encoded DNA sequences s_i and s_j, the distance measure must satisfy the condition $f_1(s_i, s_j) \geq d_{\min}$, where $d_{\min}$ is the minimum allowable distance between the sequences.
- Thermodynamic constraint (f_2): For each encoded DNA sequence s_i, the melting temperature (T_m) must lie within the predefined thresholds: $T_{\min} \leq f_2(s_i) \leq T_{\max}$, where $T_{\min}$ and $T_{\max}$ represent the minimum and maximum melting temperature limits, respectively.
- Secondary structure constraint (f_3): No encoded DNA sequence s_i should form secondary structures, which is indicated by the criterion $f_3(s_i) = 0$.
- Special subsequence constraint (f_4): No encoded DNA sequence s_i may contain specific unwanted subsequences, represented by the condition $f_4(s_i) = 0$.

For a given coding length n, the maximum coding set problem can be formulated based on these constraints as follows:

$$\begin{aligned} &\operatorname{argmax}_{S \subseteq S_n} |S| \\ &\text{s.t.} \begin{cases} f_1(s_i, s_j) \geq d_{\min}, \forall s_i, s_j \in S \\ T_{\min} \leq f_2(s_i) \leq T_{\max}, \forall s_i \in S \\ f_3(s_i) = 0, \forall s_i \in S \\ f_4(s_i) = 0, \forall s_i \in S \end{cases} \end{aligned} \tag{5.15}$$

5.2.3 *Classification of DNA Coding Algorithms*

Early research on DNA coding algorithms primarily conceptualized the DNA coding problem as a threshold-based task, which involved assessing whether specific properties of each sequence pair exceeded a predefined threshold. A range of techniques were employed in this context, including exhaustive search,

random search, dynamic programming, template mapping, graph theory methods, and statistical approaches. With advancements in the field, the DNA coding problem began to be viewed more as an optimization challenge. Consequently, heuristic algorithms emerged as essential tools for tackling this complex issue. Notably, swarm intelligence optimization algorithms and evolutionary algorithms, particularly multi-objective evolutionary optimization algorithms, have proven to be particularly effective in developing DNA sequences.

To thoroughly analyze the constraints utilized in contemporary DNA coding algorithms, we systematically summarize these constraints, highlighting their similarities and differences. Table 5.2 presents this summary, listing 14 common constraints that encompass various aspects, including (1) similarity, (2) H-measure, (3) complement Hamming distance, (4) self-complement Hamming distance, (5) completely complementary at 3'-end, (6) minimum substring, (7) Hamming distance, (8) free energy, (9) melting constraint, (10) Hairpin and other complex structures, (11) GC content, (12) continuity, (13) Base, and (14) special subsequence. These constraints are essential for DNA coding design, as they ensure that the constructed DNA sequences can stably and effectively carry out computational tasks during experimental processes while reducing the risk of interference and instability.

It is essential to recognize that the application of various constraint conditions in coding design not only emphasizes the focus of different methods on the properties of DNA sequences but also reflects distinct optimization objectives in practical applications. For example, the self-complementary Hamming distance constraint is primarily employed to mitigate self-pairing issues in sequences. In contrast, constraints related to GC content and melting temperature are directly linked to the physical stability of the DNA sequence. As a result, the selection of suitable constraint conditions often relies on specific application requirements and experimental settings. This variability also accounts for the diversity in effectiveness and efficiency observed in current DNA coding methods.

5.3 Counting DNA Coding Sequences Based on GC Content

The melting temperature of double-stranded DNA molecules is positively correlated with their GC content. Consequently, targeting a GC content of approximately 50% is advisable when designing DNA coding. This section provides some theoretical results on counting DNA coding sequences based on their GC content and introduces an algorithm for constructing DNA sequences with identical GC content.

Table 5.2 Current DNA coding algorithm classification, where DC denotes Distance Constraint, TC denotes Thermodynamic Constraint, SSC denotes Secondary Structural Constraint, SC denotes Sequence Constraint, SS denotes Search Strategy, TM denotes Template Mapping, GTM denotes Graph Theory Method, SM denotes Statistical Method, EA denotes Evolutionary Algorithm, MOA denotes Multi-Objective Optimization Algorithm, and SIA denotes Swarm Intelligence Algorithm

Method \ Constraint			DC							TC		SSC	SC			
			(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)	(13)	(14)
Threshold-based Algorithm	SS	[21]						✓	✓				✓			✓
		[22]								✓	✓			✓	✓	
		[29]								✓						
		[36]			✓				✓				✓			
	TM	[16]	✓	✓								✓	✓	✓		✓
		[17]			✓				✓	✓			✓			
		[19]	✓	✓							✓		✓			✓
	GTM	[34]		✓												
		[46] [53]						✓			✓		✓			✓
		[54]								✓						
	SM	[25]	✓	✓		✓	✓				✓		✓	✓		
Heuristic-based Algorithm	EA	[27]			✓				✓		✓	✓	✓	✓		
		[55]	✓	✓				✓								
		[56]							✓							
	MOA	[12] [40] [41] [57]	✓	✓							✓	✓	✓	✓		
		[26]	✓	✓							✓		✓	✓		
		[28]	✓	✓					✓		✓		✓	✓		
		[58]	✓	✓			✓				✓	✓	✓	✓		
		[50]	✓	✓			✓				✓	✓	✓	✓	✓	
		[59]	✓	✓						✓	✓	✓	✓	✓		
	SIA	[30] [43] [60]	✓	✓							✓	✓	✓	✓		
		[42]	✓	✓						✓	✓	✓	✓	✓		

5.3.1 Counting Theory for Designing DNA Sequences

For two single-stranded DNA sequences $x, y \in S(n)$, the hybridization distance between them, denoted as $\ell(x, y)$, refers to the number of Watson-Crick (WC) complementary base pairs between corresponding positions of x and y. It is

mathematically expressed as:

$$\ell(x, y) = \sum_{i=1}^{n} \ell(x_i, y_i) \tag{5.16}$$

where

$$\ell(x_i, y_i) = \begin{cases} 1 & \text{if } x_i \text{ and } y_i \text{ are WC complementary,} \\ 0 & \text{otherwise.} \end{cases} \tag{5.17}$$

Next, we refine the set $S(n)$: For positive integers m, λ, μ, r that are not exceed n, we define $S(n, m) \subset S(n)$ as the set of all DNA sequences of length n whose GC content (the number of bases G or C) is exactly m. Similarly, $S(n, m, \lambda) \subseteq S(n, m)$ represents the set of sequences in $S(n, m)$ such that the hybridization distance between any two sequences x and y satisfies the following four conditions:

We will now refine the set $S(n)$. For positive integers m, λ, μ, r that do not exceed n, we define the subset $S(n, m) \subset S(n)$ as the collection of all DNA sequences of length n with a GC content of exactly m bases, where the bases are either G or C. In a similar manner, we define $S(n, m, \lambda) \subseteq S(n, m)$ as the subset of sequences within $S(n, m)$ that adhere to specific criteria: the hybridization distance between any two sequences x and y must satisfy the following four conditions:

$$\begin{cases} l(x, y) \leq \lambda \\ l(x, y^r) \leq \lambda \\ l(x, y^c) \leq \lambda \\ l(x, y^{rc}) \leq \lambda \end{cases} \tag{5.18}$$

where y^r, y^c and y^{rc} are the reverse, complementary and reverse-complementary sequences of y, respectively.

Furthermore, $S(n, m, \lambda, \mu) \subseteq S(n, m, \lambda)$ denotes the set of sequences in $S(n, m, \lambda)$ for which the length of the longest common subsequence between any two sequences x and y is less than or equal to μ. Additionally, $S(n, m, \lambda, \mu, r) \subseteq S(n, m, \lambda, \mu)$ refers to the set of sequences in $S(n, m, \lambda, \mu)$ where the number of consecutive identical bases in each sequence does not exceed r.

In DNA computing and other DNA hybridization reactions, the required number of DNA sequences, denoted as N, is usually predetermined. Consequently, it is crucial to design at least N sequences that adhere to various constraints while minimizing their lengths. However, accurately calculating the number of single-stranded DNA sequences of minimum length that fulfill these constraints presents a significant challenge. We refer to this issue as the DNA coding counting problem. In the following, we focus on studying the set $S(n, m)$, providing its counting formula, and exploring the relationship between $|S(n, m)|$ and $|S(n)|$.

Theorem 5.1 *For any positive integers n and m, where* $1 \leq m \leq n$, *we have*

$$|S(n,m)| = \binom{n}{m} \times 2^n = \frac{n!}{m!(n-m)!} \times 2^n \tag{5.19}$$

Proof To calculate the total number of DNA sequences of length n that contain exactly m G and C bases (i.e., a GC content of m), we can follow a structured approach:

First, we select m positions from a total of n available positions. The number of ways to make this selection is represented by the binomial coefficient:

$$\binom{n}{m} = \frac{n!}{m!(n-m)!} \tag{5.20}$$

Next, for each arrangement of length m consisting exclusively of G and C bases, there are 2^m possible combinations of these bases.

Finally, for each configuration of G and C bases, there are 2^{n-m} possible sequences for the remaining $n-m$ positions, which can be filled with any of the two DNA bases A or T.

By combining these elements, we arrive at the total number of DNA sequences of length n that contain exactly m G and C bases.

$$\binom{n}{m} \times 2^m \times 2^{n-m} = \frac{n!}{m!(n-m)!} \times 2^m \times 2^{n-m} = \frac{n!}{m!(n-m)!} \times 2^n \tag{5.21}$$

The conclusion presented below follows from Theorem 5.1.

Corollary 5.1 *The total number of DNA sequences of length n that exhibit a GC content of* $\lfloor \frac{n}{2} \rfloor$ *is determined by the following expression:*

$$|S(n, \lfloor \frac{n}{2} \rfloor)| = \frac{n!}{\lceil \frac{n-1}{2} \rceil ! \times \lfloor \frac{n+1}{2} \rfloor !} \times 2^n \tag{5.22}$$

Corollary 5.1 provides the formula for calculating the number of DNA sequences with a GC content close to 50% of the sequence length. For example, the total number of DNA sequences with length $n = 8$ and GC content of 4 is:

$$|S(8,4)| = \frac{8!}{4! \times 4!} \times 2^8 = 70 \times 2^8 = 17920.$$

Similarly, for DNA sequences of length $n = 9$ with a GC content of 4, the total number is:

$$|S(9,4)| = \frac{9!}{4! \times 5!} \times 2^9 = 252 \times 2^9 = 64512.$$

The following lemma demonstrates that, with an increase in sequence length, the number of DNA sequences with GC content close to 50% diminishes significantly when compared to the total number of DNA sequences of equivalent length.

Lemma 5.1 *For the set $S(n, \lfloor n/2 \rfloor)$ of DNA sequences of length n with GC content of $\lfloor n/2 \rfloor$, the following holds:*

$$\lim_{n \to +\infty} \frac{|S(n, \lfloor n/2 \rfloor)|}{|S(n)|} = 0 \tag{5.23}$$

Proof According to Corollary 5.1, we have

$$\frac{|S(n, \lfloor n/2 \rfloor)|}{|S(n)|} = \frac{n!}{2^n \times \lceil (n-1)/2 \rceil ! \times \lfloor (n+1)/2 \rfloor !}$$

In the following, we prove that

$$\lim_{n \to +\infty} \frac{n!}{2^n \times \lceil (n-1)/2 \rceil ! \times \lfloor (n+1)/2 \rfloor !} = 0$$

Since

$$\begin{aligned}
&\frac{n!}{2^n \times \left\lceil \frac{n-1}{2} \right\rceil ! \times \left\lfloor \frac{n+1}{2} \right\rfloor !} \\
&= \frac{n!}{2^{\left\lceil \frac{n-1}{2} \right\rceil} \times \lceil \frac{n-1}{2} \rceil ! \times 2^{\left\lfloor \frac{n+1}{2} \right\rfloor} \times \lfloor \frac{n+1}{2} \rfloor !} \\
&= \frac{n!}{(2 \times 4 \times 6 \times \cdots \times 2 \left\lceil \frac{n-1}{2} \right\rceil) \times (2 \times 4 \times 6 \times \cdots \times (2 \left\lfloor \frac{n+1}{2} \right\rfloor - 2) \times 2 \left\lfloor \frac{n+1}{2} \right\rfloor)} \\
&= \frac{1 \times 3 \times 5 \times \cdots \times (2 \left\lfloor \frac{n-1}{2} \right\rfloor - 1) \times (2 \left\lfloor \frac{n-1}{2} \right\rfloor + 1)}{2 \times 4 \times 6 \times \cdots \times (2 \left\lfloor \frac{n+1}{2} \right\rfloor - 2) \times 2 \left\lfloor \frac{n+1}{2} \right\rfloor} \\
&= \frac{\begin{array}{l}\sqrt{1 \times 3} \times \sqrt{3 \times 5} \times \sqrt{5 \times 7} \times \cdots \\ \times \sqrt{(2 \left\lfloor \frac{n-1}{2} \right\rfloor - 1)(2 \left\lfloor \frac{n-1}{2} \right\rfloor + 1)} \times \sqrt{\left(2 \left\lfloor \frac{n-1}{2} \right\rfloor + 1\right)}\end{array}}{2 \times 4 \times 6 \times \cdots \times (2 \left\lfloor \frac{n+1}{2} \right\rfloor - 2) \times 2 \left\lfloor \frac{n+1}{2} \right\rfloor}
\end{aligned}$$

For any non-negative real numbers a and b, it holds that $\sqrt{ab} \leq \frac{a+b}{2}$ by the relationship between the geometric mean and the arithmetic mean. This inequality leads to the following results.

$$\sqrt{1 \times 3} \leq 2,$$

$$\sqrt{3 \times 5} \leq 4,$$

$$\sqrt{5 \times 7} \leq 6,$$

$$\vdots$$

$$\sqrt{(2\lfloor (n-1)/2 \rfloor - 1) \times (2\lfloor (n-1)/2 \rfloor + 1)} \leq 2\lfloor (n-1)/2 \rfloor$$

Therefore,

$$0 \leq \frac{n!}{2^n \times \lceil \frac{n-1}{2} \rceil ! \times \left\lfloor \left(\frac{n+1}{2}\right) \right\rfloor !} \tag{5.24}$$

$$\leq \frac{2 \times 4 \times 6 \times \cdots \times 2\left\lfloor \frac{n-1}{2} \right\rfloor \times \sqrt{2\left\lfloor \frac{n-1}{2} \right\rfloor + 1}}{2 \times 4 \times 6 \times \cdots \times (2\left\lfloor \frac{n+1}{2} \right\rfloor - 2) \times 2\left\lfloor \frac{n+1}{2} \right\rfloor} \tag{5.25}$$

$$= \frac{\sqrt{2\left\lfloor \frac{n-1}{2} \right\rfloor + 1}}{2\left\lfloor \frac{n+1}{2} \right\rfloor} < \frac{\sqrt{2\left\lfloor \frac{n+1}{2} \right\rfloor}}{2\left\lfloor \frac{n+1}{2} \right\rfloor} = \frac{1}{\sqrt{2\left\lfloor \frac{n+1}{2} \right\rfloor}} \to 0 \, (n \to \infty)$$

And hence,

$$\lim_{n \to +\infty} \frac{n!}{2^n \times \lceil (n-1)/2 \rceil ! \times \lfloor (n+1)/2 \rfloor !} = 0$$

The following theorem demonstrates that the conclusion of Lemma 5.1 is applicable to any DNA sequence set $S(n, m)$ characterized by length n and GC content m.

Theorem 5.2 *For the DNA sequence set $S(n, m)$, which includes DNA sequences of length n with a GC content of m (where $1 \leq m \leq n-1$), as n approaches infinity, the ratio of $|S(n, m)|$ to $|S(n)|$ tends toward zero. That is,*

$$\lim_{n \to \infty} \frac{|S(n, m)|}{|S(n)|} = 0 \tag{5.26}$$

Proof Since for any m $(1 \leq m \leq n-1)$, we have

$$\binom{n}{m} \leq \binom{n}{\lfloor n/2 \rfloor} \tag{5.27}$$

By Theorem 5.1, we know that $|S(n,m)| \leq |S(n,\lfloor n/2 \rfloor)|$. Thus, from Lemma 5.1, the conclusion holds.

The GC content of a DNA sequence influences the melting temperature T_m. To achieve consistent melting temperatures across all designed DNA sequences, it is advisable to maintain uniform GC content. In practical applications, for a given DNA sequence length n, the typical range for GC content m is as follows:

$$\left\lceil \frac{1}{3}n \right\rceil \leq m \leq \left\lceil \frac{2}{3}n \right\rceil \tag{5.28}$$

5.3.2 DNA Coding Design with Identical GC Content

Building on the concepts discussed in Sect. 5.3.1, this section introduces the construction algorithm for the set $S(n,m)$ of DNA sequences with length n and GC content m, where $1 \leq m \leq n-1$. For ease of reference, we represent the four nucleotide bases using the symbols $0, \overline{0}, 1, \overline{1}$, as follows:

$$\mathrm{A} \to 0, \quad \mathrm{T} \to \overline{0}, \quad \mathrm{C} \to 1, \quad \mathrm{G} \to \overline{1} \tag{5.29}$$

Thus, DNA sequences can be encoded using sequences consisting of $0, \overline{0}, 1, \overline{1}$. For example, the sequence AATGCTGCATGG is encoded as $00\overline{0}1\overline{1}0\overline{1}1\overline{0}0\overline{1}\overline{1}$.

We now present the specific steps and methods for constructing $S(n,m)$:

Step 1. Identify the positions of the m ones in a sequence of length n. The remaining $n-m$ positions will be filled with zeros. The total number of such combinations is given by $\binom{n}{m}$. To determine these positions, we can treat the sequence of length n as a binary number and systematically arrange the combinations in either ascending (or descending) order. For instance, when $n = 5$ and $m = 2$, the possible sequences can be listed as follows: 00011, 00101, 00110, 01001, 01010, 01100, 10001, 10010, 10100, and 11000.

Step 2. Apply the complementary operation to the m ones in each sequence identified in Step 1. The number of complementary sequences equals 2^m for every sequence. This operation is carried out by carefully arranging the sequences in ascending (or descending) order. For example, for the sequence labeled 00101, the complementary of the ones generates the following sequences: $00101, 00\overline{1}01, 00\overline{1}0\overline{1}$, and $00\overline{1}\overline{0}\overline{1}$.

Step 3. Perform the complementary operation on the $n-m$ zeros in each sequence determined in Step 2. The number of complementary sequences for

each sequence is 2^{n-m}. The complement operation is carried out in the same manner as in Step 2 by arranging the sequences in ascending (or descending) order. For example, for the sequence $00\bar{1}01$ from the five sequences mentioned earlier, the complementary operation on the zeros yields the following results: $00\bar{1}01$, $\overline{00}1\bar{0}1$, $\overline{00}101$, $\bar{0}0\bar{1}01$, $\bar{0}0\bar{1}01$, $0\overline{0}1\bar{0}1$, $0\overline{0}101$, and $00\overline{1}\bar{0}1$.

Following this procedure, all the sequences in $S(n, m)$ can be constructed.

5.4 Template Method

The template-based DNA coding method represents an early classical approach to generating DNA codes, initially proposed by Frutos et al. [16]. This method constructs a set of DNA sequences, denoted as S, by utilizing a binary template set T and a mapping set M according to the following rule:

$$T \times M \rightarrow S \tag{5.30}$$

The generation rules are defined as follows: $1 \times 1 \rightarrow$ T, $1 \times 0 \rightarrow$ A, $0 \times 1 \rightarrow$ G, and $0 \times 0 \rightarrow$ C. This method has proven effective in surface-based DNA computing for solving the SAT problem. The sequence generation rules indicate that the primary function of the template set is to dictate the positions of AT and GC pairs within a sequence. In contrast, the mapping set determines the specific nucleotide at each position—either A or T and G or C.

5.4.1 Preliminaries for Template Method

The template coding method generally requires two key conditions. (1) The GC content of the encoded sequences should be approximately 50%. (2) For any two generated sequences s_i and s_j, both the Hamming distance $H(s_i, s_j)$ and the reverse-complementary Hamming distance $H^{rc}(s_i, s_j)$ must satisfy the following criteria: $H(s_i, s_j) \geq d$ and $H^{rc}(s_i, s_j) \geq d$, to prevent complementary hybridization between the sequences, where d is typically less than half the length of the sequences. Condition (1) can be achieved easily by ensuring an equal distribution of 0s and 1s in each sequence within the template set T. To fulfill condition (2), the template set T must undergo refinement. The following method is proposed to achieve this.

For a specific template sequence $t \in T$ and a mapping sequence $m \in M$, we can define $s = t \times m$ (from $5'$ to $3'$) as the sequence generated by t and m. It is evident from the definition of template coding that the reverse sequence s^r can be obtained by reversing both the template sequence t^r and the mapping sequence m^r, i.e., $s^r = t^r \times m^r \quad (3' \rightarrow 5')$. In a similar manner, the complementary sequence s^c

can be constructed by the reverse sequence t^r of t and the reverse-complementary sequence m^{rc} of the mapping sequence m, i.e., $s^c = t^r \times m^{rc}$ $(3' \rightarrow 5')$.

In addition, if the template sequence $t \in T$ is symmetric, and the mapping sequences $m_1, m_2 \in M$ satisfy the condition $m_1^r = m_2^c$, then the sequences $s_1 = t \times m_1$ and $s_2 = t \times m_2$, generated from t using m_1 and m_2, respectively, are reverse-complementary. Building on this insight, the template set T can be categorized into two subsets: T_n (the asymmetric template set) and T_s (the symmetric template set), such that $T = T_n \cup T_s$ and $T_n \cap T_s = \emptyset$. The corresponding mapping sets for T_n and T_s are denoted as M_n and M_s, respectively. Once the subsets T_n, T_s, M_n, and M_s are established, the size of the coding set S generated through the template method is expressed by the following formula:

$$|S| = |T_n| \times |M_n| + |T_s| \times |M_s| \tag{5.31}$$

The template-based coding method has three key properties:

Property 1 If the asymmetric template set T_n fulfills the criteria for both Hamming and reverse-complementary Hamming distances, and the mapping set M_n meets the Hamming distance requirement, then the sequence set S_n generated by T_n and M_n satisfy both the Hamming and reverse-complementary Hamming distances.

Property 2 If the symmetric template set T_s satisfies only the Hamming distance requirement, and the symmetric mapping set M_s meets both the Hamming and reverse-complementary Hamming distance criteria, then the sequence set S_s generated by T_s and M_s satisfy both the Hamming and reverse complement Hamming distances.

Property 3 If both the asymmetric template set T_n and the symmetric template set T_s satisfy the Hamming distance criteria, then the sequence set $S = S_n \cup S_s$ satisfy both the Hamming and reverse-complementary Hamming distances.

Based on the above three properties, the template-based coding method can be summarized as follows: When the template sequences $t_1, t_2 \in T$ or the mapping sequences $m_1, m_2 \in M$ adhere to a specific distance constraint, the resulting DNA sequences s_1, s_2 also exhibit the corresponding properties. The fundamental idea behind this method is to convert the DNA coding problem defined over the alphabet $\Sigma_{DNA} = \{\text{A, G, C, T}\}$ into one defined over the binary alphabet $\Sigma_{01} = \{0, 1\}$. The final coding set is generated by identifying the largest asymmetric template set T_n and symmetric template set T_s, along with their respective mapping sequence sets M_n and M_s, which meet the specified conditions.

According to Property 3, it is clear that once the template sets T_n and T_s that satisfy the Hamming distance criteria are established, the sequence set S fulfilling condition (2) can be derived. This problem can be solved using graph theory. The following method describes determining the template set by locating independent sets within an n-dimensional hypercube.

To generate a DNA sequence set of length n, we aim to identify the template sets T_n and T_s that satisfy the Hamming distance criterion among the 2^n binary

sequences of length n. We begin by constructing an n-dimensional hypercube, denoted as G_n. The vertices of G_n consist of the 2^n binary sequences of length n. An edge connects two vertices if their corresponding binary sequences differ by exactly one bit, which indicates a Hamming distance of 1. As a result, each vertex in G_n is adjacent to exactly n other vertices. The sets T_n and T_s that meet the Hamming distance requirement correspond to independent sets in G_n; these are subsets of vertices where no two vertices are adjacent. For convenience, we divide the vertices of G_n into $n+1$ classes, with each class corresponding to a column. The i-th class (where $i = 1, 2, \ldots, n+1$) contains the vertices that correspond to binary sequences of length n with exactly $i - 1$ ones. It is evident that each column of G_n generates sequences with the same GC content, and sequences generated from different columns have different GC contents. To meet the template coding condition (1), which requires that the GC content of the encoded sequences is approximately 50%, we restrict the search for the template sets to the middle columns: the $\lfloor \frac{n}{2} \rfloor$-th, $(\lfloor \frac{n}{2} \rfloor - 1)$-th, and $(\lfloor \frac{n}{2} \rfloor + 1)$-th columns. In this way, the problem of determining T_n and T_s is transformed into the problem of searching for independent sets in the middle columns of G_n.

5.4.1.1 Template Coding Search Algorithms

Frutos et al. [16] employed a semi-analytical and semi-inferential approach to derive the maximum template set T and the maximum mapping set M for a coding length of $n = 8$. To build upon this finding and develop template sets T and mapping sets M for different coding lengths, a random search algorithm was introduced in [18].

(1) Template Set T

Step 1. Generate a binary string set B that consists of an equal number of 0s and 1s.

Step 2. Partition the set B into symmetric subsets B_s and asymmetric subsets B_n.

Step 3. Eliminate from B_n all binary strings $x \in B_n$ for which $H^r(x, x) < d$.

Step 4. Randomly select a binary string x from B_n, add it to the asymmetric template set T_n, and remove from B_n all binary strings whose Hamming distance to T_n is less than d.

Step 5. Continue to repeat Step 4 until B_n is empty, resulting in the final asymmetric template set T_n.

Step 6. Remove from B_s all binary strings whose Hamming distance to T_n is less than d.

Step 7. Randomly select a binary string x from B_s, add it to the symmetric template set T_s, and eliminate from B_s all binary strings whose Hamming distance to T_s is less than d.

Step 8. Continue repeating Step 7 until B_s is empty, resulting in the final symmetric template set T_s.

(2) Mapping Set M

Step 1. Randomly select a binary string x from the set B and add it to the set M_n. Subsequently, remove from B all binary strings that have a Hamming distance to M_n of less than d.

Step 2. Continue to repeat Step 1 until B is empty, resulting in the final set M_n.

Step 3. Remove from B all binary strings $x \in B$ that satisfy the condition $H^{rc}(x, x) < d$.

Step 4. Randomly select a binary string x from the set B and add it to the set M_s. Remove from B all binary strings that have a Hamming distance to M_s of less than d.

Step 5. Continue to repeat Step 4 until B is empty, resulting in the final set M_s.

Here, the Hamming distance between a binary string and a set is defined as the minimum Hamming distance between the binary string and all binary strings within the set.

It is essential to understand that for a specific coding length and Hamming distance constraint, the template and mapping sets that fulfill these criteria are not unique. In practical applications, these sets can be further refined using the H-measure proposed by Garzon [34], which enhances the coding resilience against shifting hybridization. Table 5.3 illustrates the sizes of the maximum template set T and the maximum mapping set M for coding lengths of n = 8, 12, 16, and 20. According to information-theoretic coding methods, M_n represents a specialized type of Hamming code $(4k, 8k, 2k)$, where k takes on values of 1, 2, 3, and so on. In an $n = 4k$-dimensional Hamming space, the maximum number of codes with a Hamming distance of at least $2k$ is $8k$. Excluding the two specific coding sequences 0^n and 1^n, the effective number of such codes is $8k - 2$. Additionally, since the distances between vertices in each column of the binary hypercube (see Sect. 5.4.1) are always even, the distance constraint d has practical significance only when set to an even value. It is evident that if d exceeds $2k$, the size of the template set T will noticeably diminish. Therefore, it is appropriate to consider $d = 2k$ as the upper limit for the coding distance requirement.

Table 5.3 The size of the maximum template set T and mapping set M for different coding lengths

n	8	12	16	20
T_n	6	6	12	9
T_s	2	2	6	0
M_n	14	22	30	38
M_s	5	5	13	0
S	94	142	438	342

5.4.2 Thermodynamic Stability of DNA Codes

To improve the sequence specificity of DNA molecular hybridization, Frutos et al. [16] introduced symmetric "word labels" at both ends of each generated DNA sequence. The enhanced coding structure, incorporating these word labels, is represented as 5'-GCTTvvvvvvvvTTCG-3', where "vvvvvvvv" signifies the variable DNA sequence. Hybridization experiments were conducted on a gold-coated glass surface to assess the specificity of sequences generated by the template method. The findings revealed that at 22 °C, two generated DNA sequences produced false positives. In contrast, when the temperature was raised to 37 °C, hybridization occurred only when the sequences were fully complementary. This indicates that increasing the reaction temperature can effectively mitigate the occurrence of "false positives."

5.4.3 Optimization of Template Sets

Given the significant impact of template sequences on coding quality, the optimization of template quality can be achieved from two aspects [19]: (1) the H-measure property of individual template sequences and (2) the H-measure property among different template sequences. Because DNA sequences have a directional nature, it is crucial not only to ensure that pairs of sequences maintain a sufficient shift in H-measure in the same direction but also to minimize the potential for hybridization between them. As a result, the template set must satisfy the following conditions:

- For any template sequence x, its own shift distance $h(x, x)$ satisfy:

$$h(x, x) = \min\left(\text{H-measure}(x, x^r), \text{H-measure}(x, x)\right) \geq d \tag{5.32}$$

- For any two template sequences x and y, their shift distance $h(x, y)$ satisfy:

$$h(x, y) = \min\left(\text{H-measure}(x, y^r), \text{H-measure}(x, y)\right) \geq d' \tag{5.33}$$

Here, d and d' are positive integers, and $d \geq d'$. In general, for a coding length n, $d' \leq \lfloor n/3 \rfloor$.

The influence of mapping sequences on coding quality is relatively minimal. Therefore, it is typically sufficient to ensure that any two mapping sequences x and y adhere to the following Hamming distance condition: $H(x, y) \approx \lfloor n/2 \rfloor$, where n denotes the coding length.

Given the constraints established for the template and mapping sets above, the following conditions apply to any two generated sequences $s, t \in S$.

$$\text{H-measure}(s, t) \geq d' \tag{5.34}$$

$$\text{H-measure}(s^r, t^c) \geq d' \tag{5.35}$$

Upon defining the search space for the template set (which includes the collection of binary sequences with a specified ratio of 0s to 1s), we may find a significant number of binary strings within this space that display small self-shift distances. The existence of these sequences can lead random search algorithms to select binary sequences with unfavorable shift distance characteristics, adversely affecting the candidate template set's overall shift distance properties and complicating the search process. To effectively reduce the impact of subpar sequences and improve algorithm efficiency, the search process for the template set is divided into two stages: the first stage involves filtering the initial search space, while the second stage entails conducting a random search within the filtered space [61]. The specific steps of the algorithm are as follows:

Step 1. Generate a set of binary strings B with specified 0-1 content.
Step 2. Select a binary string x from the set B and remove it from the set. If the Hamming distance $h(x, x) \geq d$, add x to the set A.
Step 3. Continue repeating Step 2 until the set B is empty.
Step 4. Construct a relation matrix M based on the set A. For any two sequences x and y in A, if the Hamming distance $h(x, y) \geq d'$, then set $M_{ij} = 1$; otherwise, set $M_{ij} = 0$. (Given that $d \geq d'$, the diagonal of the matrix will consist entirely of 1's.)
Step 5. Identify the row with the most 1's in the matrix M and its corresponding columns. If multiple rows have the same number of 1's, randomly select one. Add the corresponding binary strings to the template set T, and then remove the selected row and its associated columns, along with any columns that contain only 0's.
Step 6. Repeat Step 5 until the matrix M is empty.
Step 7. The final template set T is obtained.

This phased search strategy can significantly improve the shift distances of the binary sequences in the template set when the coding lengths are 8, 12, 16, 20, or 24.

The template method provides a customized approach to generating DNA sequences, ensuring that the resulting DNA sequences adhere to specific distance constraints. In practical applications, the relevant parameters can be defined based on the situation's specific needs. However, this coding method has a significant limitation: it does not offer control over the potential formation of secondary structures or unintended hybridizations when two encoded sequences are linked together to form a longer chain.

To enhance the quality of the final DNA sequence set S produced by the template method in practical applications, the concept of a "template frame" was introduced in [62]. Given a template set $T = \{t_1, t_2, \cdots, t_m\}$, a template frame is a sequence of length $m \times n$ formed by arranging the sequences in T in some order, where n is the length of the template sequences. An optimal template frame maximizes the total shift distance for all sequences $t_1, t_2, \ldots, t_m$. Let $P_{best} = t_1 t_2 \cdots t_m$ denote an optimal template frame. The sequence set S generated through this method can be divided into m disjoint subsets based on the template sequences utilized in their production:

$$S = S_1 \cup S_2 \cup \ldots \cup S_m,$$

where S_i $(i = 1, 2, \ldots, m)$ is the subset of sequences created by the sequence t_i along with the mapping set. Please refer to [63, 64] for further information on the coding frame.

5.5 Multi-Objective Optimization Method

Effective DNA sequence design necessitates the simultaneous consideration of multiple constraints, which can be framed as various optimization objectives. As a result, the DNA coding problem can be reformulated as a multi-objective optimization challenge involving several conflicting goals. In recent years, numerous heuristic algorithms have emerged to address the complexities of multi-objective optimization in designing DNA sequences. Notable examples include genetic algorithms [65], ant colony algorithms [66–68], particle swarm optimization [69, 70], and bee algorithms [71]. These swarm intelligence methods have demonstrated considerable advantages and promise in solving DNA coding problems effectively.

5.5.1 Optimization Model for DNA Coding Design

The DNA coding design through multi-objective optimization methods typically considers six key constraints, which act as coding objective functions. These constraints include: H-measure constraint (denoted as f_{Hm}), similarity constraint (denoted as f_S), hairpin structure constraint (denoted as f_{Ha}), continuity constraint (denoted as f_C), GC content constraint (denoted as f_{GC}), and melting temperature constraint (denoted as f_{Tm}). Building upon these constraints, the multi-objective optimization model for DNA coding design can be described as follows:

$$\text{Minimize: } F(X) = [f_{Hm}(X), f_S(X), f_{Ha}(X), f_C(X), f_{GC}(X), f_{Tm}(X)] \tag{5.36}$$

This optimization aims to identify a set of DNA sequences, referred to as X, that achieves optimal values for the specified objective functions. Specifically, we aim to find a set $X = \{x_1, x_2, \ldots, x_N\}$ consisting of N DNA sequences, each of length n. The sequences within X should simultaneously minimize six predetermined constraints. The methods for calculating these six constraints are outlined as follows:

$$f_{Hm}(X) = \sum_{i=1}^{N} \text{H-measure}(x_i)$$

$$f_S(X) = \sum_{i=1}^{N} \text{Similarity}(x_i)$$

$$f_{Ha}(X) = \sum_{i=1}^{N} \text{Hairpin}(x_i)$$

$$f_C(X) = \sum_{i=1}^{N} \text{Continuity}(x_i)$$

$$f_{GC}(X) = \sum_{i=1}^{N} \left| GC(x_i) - GC^* \right|$$

$$f_{Tm}(X) = \sum_{i=1}^{N} \left| T_m(x_i) - T_m^* \right|$$

where GC^* and T_m^* represent the specified GC content and melting temperature, respectively.

5.5.2 *Multi-Objective Evolutionary Algorithm for DNA Coding Design*

In multi-objective evolutionary optimization, the evaluation of individual superiority relies on dominance relationships. An individual u is considered to dominate another individual v if u outperforms v across all objective function values, or if u is at least as good as v in every objective function and better in at least one. This relationship is denoted as $u \preceq v$. Conversely, when u and v possess a combination of advantages and disadvantages across various objectives, i.e., u excels in some areas but underperforms in others, they are classified as mutually non-dominated. In

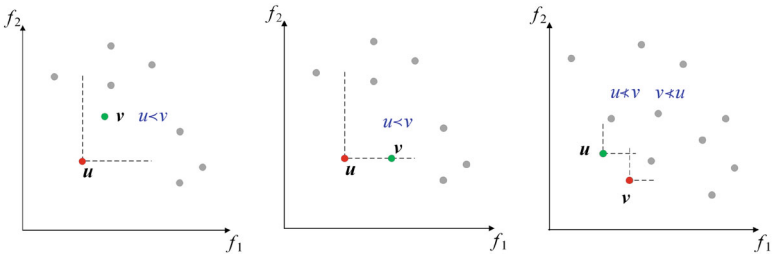


Fig. 5.2 Dominance relationships of solutions in multi-objective optimization

this scenario, a direct comparison of their superiority is not possible. Dominance relationships play a vital role in multi-objective optimization, as they facilitate the assessment of individual superiority and the construction of the Pareto front. Figure 5.2 illustrates all possible dominance relationships among individuals in multi-objective evolutionary optimization.

Comparing the superiority of mutually non-dominated candidate solutions (i.e., individuals) poses certain challenges. Nonetheless, those demonstrating a balanced performance across all objective functions are generally favored. For example, in the context of Similarity and H-measure, if the Similarity between two DNA sequences x and y is excessively high, it may lead to nonspecific hybridization between x and the complementary sequence of y. Likewise, if the H-measure values of x and y are too high, the sequences may directly complement each other, resulting in nonspecific hybridization mismatches. Thus, within the set of mutually non-dominated candidate solutions, it is optimal to select solutions that balance Similarity and H-measure. This strategy effectively minimizes nonspecific hybridization among DNA molecules, thereby enhancing the reliability of DNA computing. To facilitate this balance, the fitness function typically normalizes the values of the objective functions and incorporates a squared term that represents the difference between the Similarity and H-measure values. This approach guides the algorithm toward selecting solutions that reconcile conflicting objectives. The fitness function is defined as follows:

$$\text{Fitness}(X) = (f_S - f_{Hm})^2 + \sum_{i=1}^{m} \frac{f_i(X) - f_i^{\min}}{f_i^{\max} - f_i^{\min}} \tag{5.37}$$

Here, $f_i(X)$ represents the value of the i-th objective function for the DNA sequence set X, and m denotes the number of constrained objectives, which is typically set to 6.

5.5.3 Multi-Objective Evolutionary Algorithms for DNA Code Design

The multi-objective evolutionary algorithm is a fundamental framework for tackling multi-objective optimization problems. Its primary principle is to gradually approximate Pareto-optimal solutions through the iterative evolution of a population. Denote the population at generation t as P_t, where $P_t(i)$ represents the i-th individual in the population. Each individual corresponds to a candidate solution, that is, a set of DNA sequences. The algorithm begins by generating the population P_t and computing the objective function values for each individual p. Next, mutation operations create a new candidate solution q from the population. If $q \prec p$, then q replaces p. Conversely, if $p \prec q$, then q is discarded. In situations where neither $q \nprec p$ nor $p \nprec q$ holds, the fitness function values Fitness(q) and Fitness(p) are compared. If Fitness$(q) <$ Fitness(p), then q takes the place of p; otherwise, q is discarded. This iterative process continues, progressively approaching Pareto-optimal solutions through evolutionary iterations until the algorithm meets the predefined termination conditions.

This framework outlines the fundamental structure of evolutionary multi-objective optimization algorithms, which are extensively applied to various multi-objective optimization problems. The corresponding pseudocode is provided in Algorithm 1.

Algorithm 1 Multi-objective evolutionary algorithm framework

```
Initialization                                          ▷ Initialize the population
while t < max iteration do          ▷ Set the maximum number of evolutionary iterations
    for i = 1 to P do
        p ← P_t(i)
        q ← Mutation(P_t(i))                            ▷ Perform mutation on the individual
        Calculate Objective Functions for p and q
        if q ≺ p then                ▷ If the new individual q dominates p, replace p
            P_t(i) ← q
        else if q ⊀ p and p ⊀ q then                    ▷ If p and q are mutually non-dominated
            if Fitness(q) > Fitness(p) then
                P_t(i) ← q                              ▷ Replace p if q has better fitness
            end if
        end if
    end for
    t ← t + 1
end while
```

In recent years, a variety of algorithms for designing DNA sequences have been developed in the literature, grounded in the multi-objective evolutionary algorithm framework. These algorithms effectively achieve a balanced optimization across multiple conflicting objectives through non-dominated sorting and evolutionary mechanisms, leading to the generation of high-quality DNA sequence sets. In

2019, Chaves-González et al. [41] explored the parallelizability of multi-objective evolutionary algorithms for constrained multi-objective optimization problems, successfully producing reliable DNA sequences. The following year, Bano et al. [72] introduced an opposition-based memetic generalized differential evolution algorithm (MGDE3) to address four conflicting design criteria for reliable DNA sequence design. In 2023, Duan et al. [73] leveraged constraint functions and a block operator to simplify the dimensionality of DNA coding problems, reducing the objective functions to two and optimizing them through multi-objective evolutionary algorithms.

Given the propensity of multi-objective optimization algorithms to become ensnared in a local optimum, researchers have been exploring ways to enhance both local and global search capabilities. In 2023, Xie et al. [42] introduced a billiard-hitting strategy to adjust the positions of individuals within the population, thereby expanding the global search range. They also developed a stochastic lens opposite learning mechanism to bolster the algorithm's capacity to escape local optima. Similarly, Zhang et al. [74] proposed a two-stage constrained multi-objective evolutionary algorithm that addresses the limitations of traditional algorithms, which frequently get trapped in local optima when solving DNA coding challenges. In 2024, Wu et al. [75] proposed the LPSO algorithm by integrating improved refraction opposition-based learning and salation learning. They also introduced Gaussian mutation to increase population diversity. That same year, Zhu et al. [76] presented a Jaya algorithm based on normal clouds, which utilizes a combinatorial learning approach to update both the optimal and worst positions. This algorithm improves local search through a normal cloud model and eliminates subpar solutions using a harmony search algorithm, ultimately achieving high-quality results.

Multi-objective evolutionary optimization algorithms have exhibited remarkable effectiveness in DNA coding design. Nonetheless, critical challenges persist, particularly in enhancing their ability to address highly complex problems while ensuring generalizability across diverse experimental conditions. Furthermore, as the scale of problems expands, it is essential to optimize the computational efficiency of these algorithms to ensure their continued effectiveness.

5.6 Implicit Enumeration Method

Given a DNA coding length n, the solution space encompasses 4^n potential DNA sequences. To identify the largest DNA coding set that complies with specific constraints, the most straightforward approach involves evaluating each candidate DNA sequence individually. However, as the coding length n increases, the solution space expands exponentially, which leads to a significant rise in the computational complexity associated with this enumeration method. Balinski [77] highlighted that effectively addressing this challenge requires strategic enumeration techniques. Implementing such strategies during the enumeration process makes it possible to preemptively exclude numerous non-compliant DNA sequences, thereby

minimizing unnecessary computations. As early as 1965, Balas [78] introduced an implicit enumeration method aimed at identifying the optimal solution to a problem by examining only a subset of variable combinations. This approach has proven effective in solving 0-1 integer programming problems. Building on this foundation, reference [21] proposed a DNA coding method that integrates implicit enumeration techniques by formulating coding constraints into conditional inequalities within the framework of integer linear programming. A pruning strategy is then employed to efficiently navigate the 4^n solution space, ensuring the selection of the largest set of DNA sequences that satisfy the given constraints. This approach improves search efficiency and effectively addresses the computational challenges posed by larger problem sizes. A detailed explanation of this methodology is provided below.

5.6.1 Coding Algorithm

When generating DNA sequences of length n, we define a candidate solution $x = 5'-x_1x_2\cdots x_n-3'$, where each $x_i \in \{A, C, G, T\}$ is called a base variable. A candidate solution is deemed feasible if it complies with the specified coding constraints. The goal of DNA code design is to identify the largest possible set of feasible solutions.

The implicit enumeration coding method does not directly enumerate and evaluate all 4^n candidate solutions. Instead, it categorizes these solutions into multiple groups and carries out implicit enumeration on a group-by-group basis. To clarify how the candidate solutions are organized, we introduce the concept of a partial solution. A partial solution is defined as a sequence that incorporates both fixed bases and variable bases. For example, the sequence $5' - \mathrm{CTG}x_4x_5-3'$ serves as a partial solution of length 5, with $x_1 = \mathrm{C}$, $x_2 = \mathrm{T}$, $x_3 = \mathrm{G}$, while x_4 and x_5 represent variable bases. The complete set of a partial solution is the collection of candidate solutions generated by assigning precise bases to all base variables within the partial solution. In the aforementioned example, the complete set of the partial solution $5' - \mathrm{CTG}x_4x_5-3'$ is:

$$
\begin{array}{llll}
5'-\mathrm{CTG}\mathbf{CC}-3' & 5'-\mathrm{CTG}\mathbf{AC}-3' & 5'-\mathrm{CTG}\mathbf{TC}-3' & 5'-\mathrm{CTG}\mathbf{GC}-3' \\
5'-\mathrm{CTG}\mathbf{CA}-3' & 5'-\mathrm{CTG}\mathbf{AA}-3' & 5'-\mathrm{CTG}\mathbf{TA}-3' & 5'-\mathrm{CTG}\mathbf{GA}-3' \\
5'-\mathrm{CTG}\mathbf{CT}-3' & 5'-\mathrm{CTG}\mathbf{AT}-3' & 5'-\mathrm{CTG}\mathbf{TT}-3' & 5'-\mathrm{CTG}\mathbf{GT}-3' \\
5'-\mathrm{CTG}\mathbf{CG}-3' & 5'-\mathrm{CTG}\mathbf{AG}-3' & 5'-\mathrm{CTG}\mathbf{TG}-3' & 5'-\mathrm{CTG}\mathbf{GG}-3'
\end{array}
$$

When a partial solution x of length n incorporates $n - m$ base variables, it can generate 4^{n-m} distinct candidate solutions. Thus, the complete set derived from x comprises 4^{n-m} candidate solutions. Notably, if the partial solution x contains no base variables, the complete set will consist of just one candidate solution: x itself.

The implicit enumeration coding approach begins by generating a candidate solution and subsequently testing each partial solution against coding constraints

while considering the complete set of potential partial solutions. The process starts with a partial solution represented as $x = 5'-x_1x_2\cdots x_n-3'$, consisting of n base variables. The algorithm iteratively reduces the number of base variables by assigning specific bases to them. At each stage, a new partial solution is produced with one fewer base variable, which is then tested against the coding constraints. If all base variables are assigned and the resulting solution satisfies all constraints, the candidate solution is considered feasible. Conversely, if any partial solution violates the coding constraints, the entire set of candidate solutions associated with that partial solution is discarded, as none can be valid. This method enables the algorithm to eliminate infeasible partial solutions and avoids unnecessary further exploration based on them.

The implicit enumeration search algorithm is comprised of four key steps:

Step 1. Parameter Initialization: Start by defining the relevant parameters, including coding length, the number of codes, and various coding constraints, tailored to the scale of the specific DNA computing problem.
Step 2. Generation strategy for new candidate solutions: Generate optimal and feasible solutions by exploring all possible coding combinations. During this step, prioritize constraint parameters that exhibit the fastest convergence speed among multiple target constraints to create new candidate solutions.
Step 3. Constraint evaluation for candidate solutions: Assess each generated candidate DNA sequence to determine whether it adheres to the specific DNA coding constraints. If a candidate sequence fails to satisfy a constraint, convert the violated constraint into a new generation parameter, allowing for the production of new candidate solutions. This approach reduces the number of enumerations and enhances the algorithm's convergence speed.
Step 4. Algorithm termination rule: The algorithm stops either when all 4^n possible codes have been explored or when a sufficient number of DNA sequences that fulfill the requirements have been generated. Each time a new candidate solution is created, it is methodically checked to ensure it meets all coding constraints. If the candidate sequence fulfills all requirements, it is added to the DNA sequence set S. However, if it fails to comply with any constraint, the violated constraint is transformed into parameters for generating new candidate solutions, and the constraint testing process is repeated. This process continues until all conditions are satisfied or all potential solution spaces have been explored. The algorithm stops when the number of generated DNA sequences reaches the predetermined requirement, or when the solution space has been thoroughly searched.

5.6.2 Application of Implicit Enumeration Coding Method

Compared to methods such as the template method, genetic algorithms, simulated annealing, and multi-objective evolutionary algorithms, the DNA sequences generated through the implicit enumeration coding method demonstrate superior

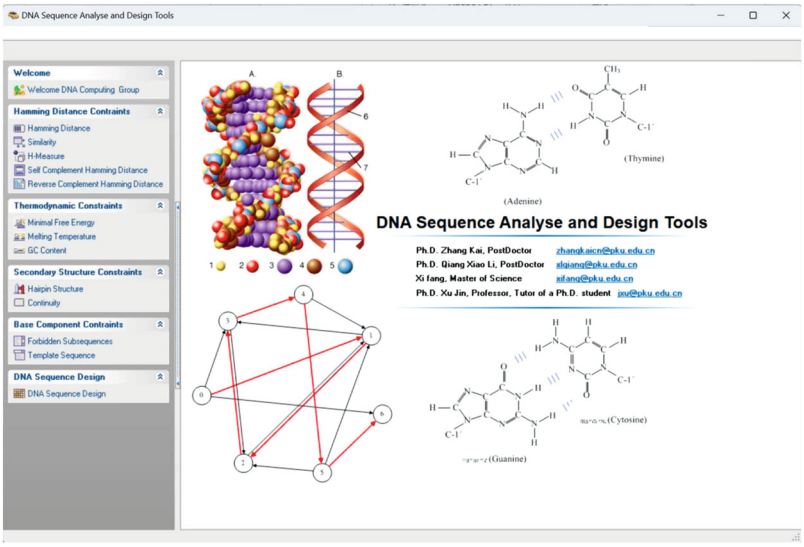


Fig. 5.3 A DNA coding analysis and design system designed by utilizing implicit enumeration techniques

stability and reliability regarding DNA coding quality. Furthermore, this approach provides improved scalability. The algorithm can generate a substantial array of DNA sequences that adhere to coding constraints, making it particularly well-suited for the design requirements of DNA molecules in various nucleic acid experiments [21].

We developed a DNA coding analysis and design software system grounded in the implicit enumeration algorithm, as depicted in Fig. 5.3. This system is designed to create DNA sequences that fulfill specific constraints tailored to various coding requirements and offers performance analysis for each generated DNA sequence. Depending on their experimental objectives, users can select from various constraints, including thermodynamics, Hamming distance, secondary structure, and base composition. The software adeptly navigates the 4^n solution space to yield the largest possible set of DNA molecules that meet the selected criteria.

The graph coloring problem is a well-established **NP**-complete problem with applications across various domains, including class scheduling, circuit layout, and register allocation. In our examination of the 3-coloring problem on a graph with 61 vertices, as depicted in Fig. 5.4, we generated 129 DNA sequences using our DNA coding analysis and design system (illustrated in Fig. 5.3) for biochemical experiments. All sequences were designed to meet specific coding constraints: 50% GC content, a maximum of three consecutive identical bases, a minimum of six overlapping subsequences, complete complementarity at the 3' end, restrictions on hairpin structures, and specific melting temperature requirements. The 129 generated DNA sequences were analyzed using the ABI DNA Analyzer. The results

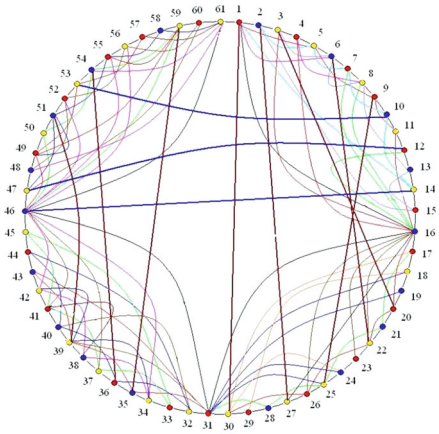


Fig. 5.4 A graph with 61 vertices [8]

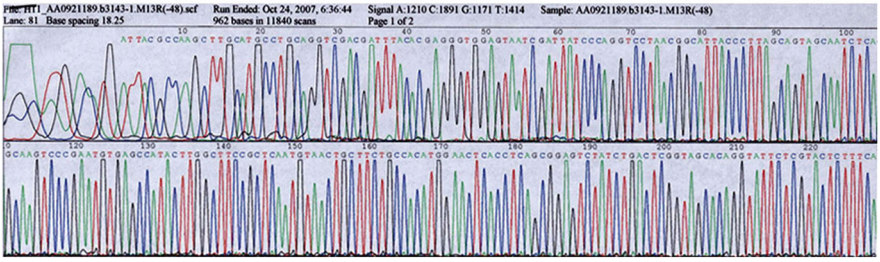


Fig. 5.5 DNA sequence analysis by ABI DNA Analyzer

demonstrated that these sequences effectively prevent non-specific hybridization between DNA molecules, as shown in Fig. 5.5. Utilizing these codes, we created a DNA-based model to determine a 3-coloring of the graph. Our biochemical experiments successfully yielded the DNA sequence representing the graph's unique solution. For additional details regarding the design of the DNA sequence and the specific DNA computation model, please refer to reference [7].

References

1. Xu J., Liu W., Zhang K., Zhu E.: DNA coding theory and algorithms. Artificial Intelligence Review **58**, 178 (2025). https://doi.org/10.1007/s10462-025-11132-x
2. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(5187), 1021–1024 (1994).
3. Lipton, R.J.: DNA solution of hard computational problems. Science **268**(5210), 542–545 (1995).
4. Braich, R.S., Chelyapov, N., Johnson, C., et al.: Solution of a 20-variable 3-SAT problem on a DNA computer. Science **296**(5567), 499–502 (2002).

5. Liu, W., Gao, L., Liu, X., et al.: Solving the 3-SAT problem based on DNA computing. Journal of Chemical Information and Computer Sciences **43**(6), 1872–1875 (2003).
6. Zimmermann, K.-H.: Efficient DNA sticker algorithms for NP-complete graph problems. Computer Physics Communications **144**(3), 297–309 (2002).
7. Xu, J., Qiang, X., Zhang, K., et al.: A DNA computing model for the graph vertex coloring problem based on a probe graph. Engineering **4**(1), 61–77 (2018).
8. Zhu, E., Luo, X., Liu, C., et al.: An operational DNA strand displacement encryption approach. Nanomaterials **12**(5), 877 (2022).
9. Liu, C., Liu, Y., Zhu, E., et al.: Cross-inhibitor: a time-sensitive molecular circuit based on DNA strand displacement. Nucleic Acids Research **48**(19), 10691–10701 (2020).
10. Wang, F., Zhang, X., Chen, X., et al.: Priority encoder based on DNA strand displacement. Chinese Journal of Electronics **33**(6), 1538–1544 (2024).
11. Garzon, M.H.: DNA codeword design: Theory and applications. Parallel Processing Letters **24**(2), 1440001 (2014).
12. Chaves-González, J.M., Vega-Rodríguez, M.A.: A multiobjective approach based on the behavior of fireflies to generate reliable DNA sequences for molecular computing. Applied Mathematics and Computation **227**, 291–308 (2014).
13. Deaton, R.J., Murphy, R.C., Garzon, M.H., et al.: Good encodings for DNA-based solutions to combinatorial problems. In: Proceedings of the DNA Based Computers, pp. 247–258. Amer Mathematical Society, New Jersey, USA (1996).
14. Deaton, R., Garzon, M., Murphy, R., et al.: Reliability and efficiency of a DNA-based computation. Physical Review Letters **80**(2), 417 (1998).
15. Deaton, R., Chen, J., Bi, H., et al.: A PCR-based protocol for in vitro selection of non-crosshybridizing oligonucleotides. In: Proceedings of the 8th International Workshop on DNA-Based Computers, pp. 196–204. Springer Berlin Heidelberg, Sapporo, Japan (2003).
16. Frutos, A.G., Liu, Q., Thiel, A.J., et al.: Demonstration of a word design strategy for DNA computing on surfaces. Nucleic Acids Research **25**(23), 4748–4757 (1997).
17. Marathe, A., Condon, A.E., Corn, R.M.: On combinatorial DNA word design. Journal of Computational Biology **8**(3), 201–219 (2001).
18. Liu, W., Wang, S., Gao, L., et al.: DNA sequence design based on template strategy. Journal of Chemical Information and Computer Sciences **43**(6), 2014–2018 (2003).
19. Arita, M., Kobayashi, S.: DNA sequence design using templates. New Generation Computing **20**, 263–277 (2002).
20. Hartemink, A.J., Gifford, D.K., Khodor, J.: Automated constraint-based nucleotide sequence selection for DNA computation. Biosystems **52**(1–3), 227–235 (1999).
21. Kai, Z., Linqiang, P., Jin, X.: A global heuristically search algorithm for DNA encoding. Progress in Natural Science **17**(6), 745–749 (2007).
22. Penchovsky, R., Ackermann, J.: DNA library design for molecular computation. Journal of Computational Biology **10**(2), 215–229 (2003).
23. Zhu, E., Jiang, F., Liu, C., et al.: Partition independent set and reduction-based approach for partition coloring problem. IEEE Transactions on Cybernetics **52**(6), 4960–4969 (2020).
24. Zhu, E., Zhang, Y., Wang, S., et al.: A dual-mode local search algorithm for solving the minimum dominating set problem. Knowledge-Based Systems **298**, 111950 (2024).
25. Tanaka, F., Nakatsugawa, M., Yamamoto, M., et al.: Developing support system for sequence design in DNA computing. In: Proceedings of the 7th International Workshop on DNA-Based Computers, pp. 129–137. Springer Berlin Heidelberg, Tampa, USA (2002).
26. Guangzhao, C., Yunyun, N., Yanfeng, W., et al.: A new approach based on PSO algorithm to find good computational encoding sequences. Progress in Natural Science **17**(6), 712–716 (2007).
27. Wei, W., Xuedong, Z., Qiang, Z., et al.: The optimization of DNA encodings based on GA/SA algorithms. Progress in Natural Science **17**(6), 739–744 (2007).
28. Jianhua, X., Jin, X., Xiutang, G., et al.: Multi-objective carrier chaotic evolutionary algorithm for DNA sequences design. Progress in Natural Science **17**(12), 1515–1520 (2007).

29. Kawashimo, S., Ono, H., Sadakane, K., et al.: Dynamic neighborhood searches for thermodynamically designing DNA sequence. In: Proceedings of the 13th International Meeting on DNA Computing, pp. 130–139. Springer Berlin Heidelberg, Memphis, USA (2008).
30. Kurniawan, T.B., Khalid, N.K., Ibrahim, Z., et al.: An ant colony system for DNA sequence design based on thermodynamics. In: Proceedings of the 4th IASTED International Conference on Advances in Computer Science and Technology, pp. 144–149. ACTA Press, Langkawi, Malaysia (2008).
31. Caserta, M., Voß, S.: A math-heuristic algorithm for the DNA sequencing problem. In: Proceedings of the International Conference on Learning and Intelligent Optimization, pp. 25–36. Springer Berlin Heidelberg, Venice, Italy (2010).
32. Fouilhoux, P., Mahjoub, A.R.: Solving VLSI design and DNA sequencing problems using bipartization of graphs. Computational Optimization and Applications **51**, 749–781 (2012).
33. Caserta, M., Voß, S.: A hybrid algorithm for the DNA sequencing problem. Discrete Applied Mathematics **163**, 87–99 (2014).
34. Garzon, M., Neathery, P., Deaton, R., et al.: A new metric for DNA computing. In: Proceedings of the 2nd Genetic Programming Conference, pp. 636–638. Morgan Kaufman, Stanford, USA (1997).
35. Feldkamp, U., Banzhaf, W., Rauhe, H.: A DNA sequence compiler. In: Proceedings of the 6th International Meeting on DNA-Based Computers, pp. 1–10. Springer Berlin Heidelberg, Leiden, Netherlands (2000).
36. Tulpan, D.C., Hoos, H.H., Condon, A.E.: Stochastic local search algorithms for DNA word design. In: Proceedings of the 8th International Workshop on DNA-Based Computers, pp. 229–241. Springer Berlin Heidelberg, Sapporo, Japan (2003).
37. Li, X., Wang, B., Lv, H., et al.: Constraining DNA sequences with a triplet-bases unpaired. IEEE Transactions on Nanobioscience **19**(2), 299–307 (2020).
38. Li, X., Wei, Z., Wang, B., et al.: Stable DNA sequence over close-ending and pairing sequences constraint. Frontiers in Genetics **12**, 644484 (2021).
39. Tanaka, F., Kameda, A., Yamamoto, M., et al.: Design of nucleic acid sequences for DNA computing based on a thermodynamic approach. Nucleic Acids Research **33**(3), 903–911 (2005).
40. Shin, S.-Y., Lee, I.-H., Kim, D., et al.: Multiobjective evolutionary optimization of DNA sequences for reliable DNA computing. IEEE Transactions on Evolutionary Computation **9**(2), 143–158 (2005).
41. Chaves-González, J.M., Martínez-Gil, J.: An efficient design for a multi-objective evolutionary algorithm to generate DNA libraries suitable for computation. Interdisciplinary Sciences: Computational Life Sciences **11**, 542–558 (2019).
42. Xie, L., Wang, S., Zhu, D., et al.: DNA sequence optimization design of arithmetic optimization algorithm based on billiard hitting strategy. Interdisciplinary Sciences: Computational Life Sciences **15**(2), 231–248 (2023).
43. Yang, X., Zhu, D., Yang, C., et al.: H-ACO with consecutive bases pairing constraint for designing DNA sequences. Interdisciplinary Sciences: Computational Life Sciences, 1–15 (2024).
44. Deaton, R., Garzon, M.: Thermodynamic constraints on DNA-based computing. Computing with Bio-Molecules, 138–152 (1998).
45. Zhu, X., Liu, W., Sun, C.: Research on the DNA words and algorithm. ACTA Electronica Sinica **34**(7), 1169 (2006).
46. Feldkamp, U., Rauhe, H., Banzhaf, W.: Software tools for DNA sequence design. Genetic Programming and Evolvable Machines **4**, 153–171 (2003).
47. SantaLucia Jr, J.: A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proceedings of the National Academy of Sciences **95**(4), 1460–1465 (1998).
48. SantaLucia Jr, J., Hicks, D.: The thermodynamics of DNA structural motifs. Annual Review of Biophysics and Biomolecular Structure **33**(1), 415–440 (2004).

49. Arita, M., Nishikawa, A., Hagiya, M., et al.: Improving sequence design for DNA computing. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation, pp. 875–882. Morgan Kaufmann Publishers, Las Vegas, USA (2000).
50. Kim, D., Soo-Yong, S., In-Hee, L., et al.: NACST/SEQ: A sequence design system with multiobjective optimization. In: Proceedings of the 8th International Workshop on DNA-Based Computers, pp. 242–251. Springer Berlin Heidelberg, Sapporo, Japan (2003).
51. Braich, R.S., Johnson, C., Rothemund, P.W., et al.: Solution of a satisfiability problem on a gel-based DNA computer. In: Proceedings of the 6th International Workshop on DNA-Based Computers, pp. 27–42. Springer Berlin Heidelberg, Leiden, Netherlands (2001).
52. Faulhammer, D., Cukras, A.R., Lipton, R.J., et al.: RNA solutions to chess problems. Proceedings of the National Academy of Sciences **97**(4), 1385–1389 (2000).
53. Feldkamp, U., Saghafi, S., Banzhaf, W., et al.: DNASequenceGenerator: A program for the construction of DNA sequences. In: Proceedings of the 7th International Workshop on DNA-Based Computers, pp. 23–32. Springer Berlin Heidelberg, Tampa, USA (2002).
54. Deaton, R., Chen, J., Bi, H., et al.: A software tool for generating non-crosshybridizing libraries of DNA oligonucleotides. In: Proceedings of the International Workshop on DNA-Based Computers, pp. 252–261. Springer Berlin Heidelberg, Sapporo, Japan (2002).
55. Zhang, B.-T., Shin, S.-Y.: Molecular algorithms for efficient and reliable DNA computing. Genetic Programming **98**, 735–742 (1998).
56. Deaton, R., Garzon, M., Murphy, R., et al.: Genetic search of reliable encodings for DNA-based computation. In: Proceedings of the First Annual Conference on Genetic Programming, pp. 421–427. MIT Press, Stanford, USA (1996).
57. Yang, G., Wang, B., Zheng, X., et al.: IWO algorithm based on niche crowding for DNA sequence design. Interdisciplinary Sciences: Computational Life Sciences **9**, 341–349 (2017).
58. Shin, S.-Y., Kim, D.-M., Lee, I.-H., et al.: Evolutionary sequence generation for reliable DNA computing. In: Proceedings of the 2002 Congress on Evolutionary Computation, pp. 79–84. IEEE, Honolulu, USA (2002).
59. Xiao, J., Jiang, Y., He, J., et al.: A dynamic membrane evolutionary algorithm for solving DNA sequences design with minimum free energy. MATCH Commun. Math. Comput. Chem. **70**(3), 971–986 (2013).
60. Khalid, N.K., Kurniawan, T.B., Ibrahim, Z., et al.: A model to optimize DNA sequences based on particle swarm optimization. In: Proceedings of the 2008 Second Asia International Conference on Modelling and Simulation, pp. 534–539. IEEE, Kuala Lumpur, Malaysia (2008).
61. Reif, J.H., LaBean, T.H., Pirrung, M., et al.: Experimental construction of very large scale DNA databases with associative search capability. In: Proceedings of the 7th International Workshop on DNA-Based Computers, pp. 231–247. Springer Berlin Heidelberg, Tampa, USA (2002).
62. Liu, W., Chen, L., Bai, B., et al.: Research on optimizing the template frame in DNA computing. ACTA Electronica Sinica **35**(8), 1490–1494 (2007).
63. Liu, W., Zhu, X., Wang, X., et al.: A new method to optimize the template set in DNA computing. Journal of Electronics and Information Technology **30**(5), 1131–1135 (2008).
64. Wang, X., Liu, W., Zhu, X., et al.: Improving the single template method in DNA computing. ACTA Electronica Sinica **37**(12), 2720–2724 (2009).
65. Wu, J.-S., Lee, C., Wu, C.-C., et al.: Primer design using genetic algorithm. Bioinformatics **20**(11), 1710–1717 (2004).
66. Kurniawan, T.B., Ibrahim, Z., Khalid, N.K., et al.: A population-based ant colony optimization approach for DNA sequence optimization. In: Proceedings of the 2009 Third Asia International Conference on Modelling and Simulation, pp. 246–251. IEEE, Bandung and Bali, Indonesia (2009).
67. Mustaza, S.M., Abidin, A.F.Z., Ibrahim, Z., et al.: A modified computational model of ant colony system in DNA sequence design. In: Proceedings of the 2011 IEEE Student Conference on Research and Development, pp. 169–173. IEEE, Cyberjaya, Malaysia (2011).

68. Yakop, F., Ibrahim, Z., Abidin, A.F.Z., et al.: An ant colony system for solving DNA sequence design problem in DNA computing. International Journal of Innovative Computing, Information and Control **8**(10), 7329–7339 (2012).
69. Zhu, D., Huang, Z., Liao, S., et al.: Improved bare bones particle swarm optimization for DNA sequence design. IEEE Transactions on Nanobioscience **22**(3), 603–613 (2022).
70. Zhang, W., Zhu, D., Huang, Z., et al.: Improved multi-strategy matrix particle swarm optimization for DNA sequence design. Electronics **12**(3), 547 (2023).
71. Chaves-González, J.M., Vega-Rodríguez, M.A., Granado-Criado, J.M.: A multiobjective swarm intelligence approach based on artificial bee colony for reliable DNA sequence design. Engineering Applications of Artificial Intelligence **26**(9), 2045–2057 (2013).
72. Bano, S., Bashir, M., Younas, I.: A many-objective memetic generalized differential evolution algorithm for DNA sequence design. IEEE Access **8**, 222684–222699 (2020).
73. Duan, H., Zhang, K., Zhang, X.: A DNA coding design based on multi-objective evolutionary algorithm with constraint. In: Proceedings of the 2023 7th International Conference on Machine Learning and Soft Computing, pp. 40–45. IEEE, Chongqing, China (2023).
74. Zhang, X., Zhang, K., Wu, N., et al.: A two-stage constrained multi-objective evolutionary algorithm for DNA encoding problem. In: Proceedings of the 2023 IEEE International Conference on Systems, Man, and Cybernetics, pp. 548–555. IEEE, Maui, USA (2023).
75. Wu, H., Zhu, D., Huang, Z., et al.: Enhanced DNA sequence design with learning PSO. Evolutionary Intelligence, 1–15 (2024).
76. Zhu, D., Wang, S., Huang, Z., et al.: A Jaya algorithm based on normal clouds for DNA sequence optimization. Cluster Computing **27**(2), 2133–2149 (2024).
77. Balinski, M.L.: Integer programming: methods, uses, computations. Management Science **12**(3), 253–313 (1965).
78. Balas, E.: An additive algorithm for solving linear programs with zero-one variables. Operations Research **13**(4), 517–546 (1965).

Chapter 6
Enumerative DNA Computing Model

From 1994 to 2004, research in DNA computing was in its nascent stage, encompassing various aspects such as computational models, encoding, experimentation, and detection. Notably, during this period, the computational models were primarily enumerative in nature. These pioneering research outcomes not only laid the foundation for the deeper exploration of DNA computing but also provided a solid groundwork for RNA computing and, more broadly, the entire field of biological computing. This chapter focuses on the enumerative DNA computing models, selecting a subset of representative achievements for detailed introduction and in-depth analysis.

6.1 DNA Computing Model for the Directed Hamiltonian Path Problem

Let G be a directed graph, v_1 and v_2 be two vertices of G. If there exists a directed path P from v_1 to v_2 that visits every other vertex in G exactly once, then P is called a directed Hamiltonian path from v_1 to v_2. This definition is analogous to the undirected Hamiltonian path in an undirected graph. It is well-known that finding the directed Hamiltonian Path Problem (HPP) for a (directed) graph is a challenging NP-complete problem. In 1994, Adleman first proposed a DNA computing model to solve the directed HPP [1]. The basic idea of this model is outlined as follows.

1. A random oligonucleotide of length 20 is assigned to each vertex of the directed graph. Based on this, a specific oligonucleotide of length 20 is associated with each directed edge in the graph.
2. A suitable amount of ligase is added, and PCR amplification is employed to obtain all directed paths from the starting point to the endpoint.
3. Electrophoresis is used to detect the required directed Hamiltonian path.

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_6

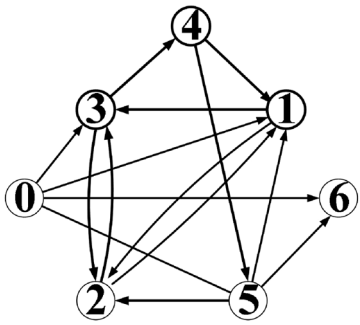


Fig. 6.1 The 7-order directed graph used in Adleman's experiment

Adleman conducted experimental verification on the directed graph shown in Fig. 6.1. The experiment lasted for a week and successfully determined the Hamiltonian path from the specified starting point "0" to the endpoint "6". This pioneering achievement marked the beginning of a new era in computation, where DNA molecules were used to perform calculations through biochemical reactions. A brief overview of Adleman's algorithm steps and the corresponding biochemical experiments is presented below.

Without loss of generality, the theoretical algorithm steps for solving the HPP on an n-order graph are as follows:

1. Randomly generate paths through the graph;
2. Only retain paths with v_{in} as the starting point and v_{out} as the end point;
3. Only retain paths with n vertices;
4. Only retain paths that visit every vertex in the graph at least once;
5. If a path satisfying all the above conditions exists, output "YES"; otherwise, output "NO".

For each step of the above algorithm, Adleman accomplished the process through biochemical reactions, as detailed below:

1. Biochemical experimental operation for the first step of the algorithm:

   (a) Assign each vertex i in the graph a random oligonucleotide $S(i)$ of length 20. The direction of these oligonucleotide fragments is $5' \rightarrow 3'$. The Watson-Crick complementary sequence of $S(i)$ is denoted as $\overline{S(i)}$. For example:

   $S(2)$: TATCGGATCG GTATATCCGA
   $S(3)$: GCTATTCGAG CTTAAAGCTA
   $\overline{S(4)}$: GGCTAGGTAC CAGCATGCTT
   $\overline{S(3)}$: TAGCTTTAAG CTCGAATAGC

   (b) For each arc $i \rightarrow j$ in the graph, construct an oligonucleotide $S(i \rightarrow j)$. The method is as follows: Divide the sequence of $S(i)$ into two parts $S_1(i)$ and $S_2(i)$, where $S_1(i)$ is the first 10 bases of $S(i)$, and $S_2(i)$ is the last 10 base sequence of $S(i)$.

Based on the above design, Adleman encoded each directed edge $i \rightarrow j$ in Fig. 6.1 into the corresponding oligonucleotide sequence $S(i \rightarrow j) = S_2(i)S_1(j)$. For example,

$$S(2 \rightarrow 3) = \text{GTATATCCGA GCTATTCGAG}$$
$$S(3 \rightarrow 2) = \text{CTTAAAGCTA TATCGGATCG}$$
$$S(3 \rightarrow 4) = \text{CTTAAAGCTA GGCTAGGTAC}$$

(c) Generate paths. For each vertex and each arc in the graph, in a single ligation reaction, take out 50 pmol of $\overline{S(i)}$ and 50 pmol of S(i $\rightarrow$ j) and mix them together; $\overline{S(i)}$ Nucleotides act as splints, connecting compatible arcs together, so the final result of the ligation reaction leads to the production of DNA molecules representing random paths of the corresponding graph.

$$S(2 \rightarrow 3) \qquad S(3 \rightarrow 4)$$

$$\downarrow$$

$$5'\text{GTATATCCGA GCTATTCGAGCTTAAAGCTA GGCTAGGTAC } 3'$$
$$3'\text{CGATAAGCTCGAATTTCGAT } 5'$$

$$\overline{S(3)}$$

(d) For each directed edge in the graph, approximately 3×10^{13} copies (about 50 pmol) are required to be added to the ligation reaction. Therefore, if a Hamiltonian path exists, DNA molecules encoding the directed Hamiltonian path will inevitably be generated.

2. Biochemical experimental operation for the second step of the algorithm:
   Using the product of the first step as a template, PCR amplification is performed using $S(0)$ and $\overline{S(6)}$ as primers. The resulting DNA molecules represent all path sets starting from 0 and ending at 6.
3. Biochemical experimental operation for the third step of the algorithm:
   The product of the second step is subjected to gel filtration chromatography, and all DNA molecules containing 140 base pairs are separated, which represents paths with exactly 7 vertices. Soak these DNA molecules in double-distilled water ($dd H_2O$) for extraction. The resulting product is amplified by PCR, filtered by agarose gel, and purified several times to further increase the purity and proceed to the next step.
4. Biochemical experimental operation for the fourth step of the algorithm:
   The product of the third step is subjected to affinity chromatography using a magnetic bead separation system. First, the double-stranded DNA is denatured and unwound into various single-stranded DNA, and then the single-stranded DNA in the test tube is incubated with $\overline{S(1)}$ bound to magnetic beads. Only those single-stranded DNA molecules that contain the sequence $S(1)$, which encodes

vertex 1 at least once, are retained. Subsequently, the above steps are repeated in sequence with $\overline{S(2)}$, $\overline{S(3)}$, $\overline{S(4)}$, and $\overline{S(5)}$ bound to magnetic beads.

5. Biochemical experimental operation for the fifth step of the algorithm:
   The product of the fourth step is used as a substrate for PCR amplification and gel electrophoresis analysis, and DNA chains containing all vertices are screened out to obtain the solution to the problem.

The above content details the DNA computing model created by Adleman for solving the directed Hamiltonian path problem of directed graphs. Based on this model, our research group has further established a DNA computing model for the weighted Hamiltonian path problem (including both directed and undirected types). This will not be detailed here, interested readers can refer to Reference [19].

6.2 DNA Computing Model of Satisfiability Problem

Definition 6.1 Suppose $A = \{x_1, x_2, L, x_n\}$is a set of Boole variables. A clause is a formula $C = b_1 \vee b_2 \vee \cdots \vee b_k$. The symbol $\vee$ represents the logic "or", and each b_i is a variable in A or the negation of a variable in A. The negation of the element x_i in A is represented by the symbol $\overline{x_i}$. A propositional formula can be seen as a formula composed of the "and" of several clauses: $F = C_1 \wedge C_2 \wedge \cdots \wedge C_r$. Each C_i is a clause, and the symbol $\wedge$ represents the logic "and".

The satisfiability problem aims to find the set of solutions that satisfy the logical formula $F = 1$. Obviously, this problem is an NP-complete problem. Currently, there are many models for SAT (satisfiability problem) based on DNA computing. Here, we will focus on the work achievements made by Lipton [16] in 1995 and Braich et al. [3] in 2002. For other related models, please refer to Reference [18].

In 1995, Lipton [16] imitated Adleman's method and gave a DNA computing model for the satisfiability problem (SAT problem). The basic idea of this model is: let each feasible solution of the SAT problem correspond to an n bit binary number, construct a simple contact network G_n, and correspond the n-bit binary data pool to the directed path from the starting point a_1 to the end point a_{n+1} in the network G_n. Then, construct a DNA data pool using the "Watson-Crick" method, and use four basic molecular biology technologies to obtain all solutions to the SAT problem.

Lipton's basic idea is as follows:

1. If a formula $F = C_1 \wedge C_2 \wedge \cdots \wedge C_r$ contains k variables, then all k- bit DNA molecule strings are formed and put into a test tube t_0;
2. For each clause $C = b_1 \vee b_2 \vee \cdots \vee b_m$ ($m \leq k$, $i = 1, 2, \ldots, r$), and $j = 1, 2, \ldots, k$, extract the k bits where the b_j-bit is 1 from the test tube t_{i-1} (that is, if $b_j = x$, then the value of the b_j bit is 1; if $b_j = \overline{x}$, then the value of the b_j -bit is 0). Combine all clauses in the test tube t_i;

3. If there is DNA in the final test tube t_r, the answer is "YES", otherwise it is "NO".

Next, we will give a more detailed description of Lipton's specific calculation steps.

1. The vertices and directed edges are encoded in the same way as Adleman: that is, both vertices and edges are 20 bp in length. For any vertex i, it is represented by $p_i q_i$, where p_i and q_i are the sequences of the first 10 bp and the last 10 bp of the DNA molecule representing vertex i; for any directed edge $i \rightarrow j$, it is represented by $\overline{p}_i \overline{q}_i$. The DNA molecules encoding vertices and edges are put into the initial test tube t_0. After sufficient reaction, DNA molecules representing various directed paths in the graph G will be formed.
2. Using p_{a1} and $\overline{q_{a3}}$ as primers to search for directed paths starting with a_1 and ending with a_3 in the test tube t_0, so there are only DNA molecules representing the above 4 paths left in t_0.
3. Search for the DNA molecule with the first bit as 1 ($x = 1$) from test tube t_0 and put it into test tube t_1, the rest are put into test tube t_1'; then search for the DNA molecule with the second bit as 1 (y = 1) from test tube t_1' and put it into test tube t_2; combine test tubes t_1 and t_2 into t_3 to get the DNA molecules that satisfy the first clause.
4. Search for the DNA molecule with the first bit as 0 ($\overline{x} = 1$) from test tube t_3 and put it into test tube t_4, the rest are put into test tube t_4'; then search for the DNA molecule with the second bit as 0($\overline{y} = 1$) from test tube t_4' and put it into test tube t_5; combine test tubes t_4 and t_5 into t_6 to get the DNA molecules that satisfy the second clause.
5. Check test tube t_6, if there are DNA molecules, they are the solution to the SAT problem; otherwise, the problem has no solution.

Lipton's main contribution is that he translated the base pairs on the DNA molecule into a string of 1s and 0s. Then, he took the DNA molecules with predetermined sequences from different test tubes and mixed them, enabling them to mimic electronic gates for making 'yes' or 'no' judgments. In other words, he gave DNA the ability to "think" and make logical judgments. Recently, Faulhammer et al. [8] implemented Lipton's design using biotechnology in experiments.

In 1999, Landweber et al. [6] proposed a "destructive" SAT problem algorithm based on RNA by using the property of RNA enzyme H to specifically recognize and hydrolyze RNA sequences that are completely complementary to DNA. This algorithm operates by using RNA enzyme H to eliminate solutions in the search space that do not satisfy the given conditions. In 2000, Sakamoto et al. [23] from Tokyo University cleverly used the "hairpin" structure of single-stranded DNA molecules to encode the constraints of logical operations in DNA molecules, and solved a 3-SAT problem through the self-assembly process of DNA molecules (i.e., forming a "hairpin" structure). In 2003, Liu Wenbin et al. proposed a new algorithm for solving 3-SAT problems based on DNA computing on the basis of

Sakamoto's research [18]. The improved algorithm they proposed based on the text string strategy has the following advantages:

1. The maximum number of text strings generated during the calculation process is greatly reduced;
2. The length of the text string is reduced;
3. The main operation of this algorithm is extraction and synthesis.

In 2000, Liu et al. [17] proposed a algorithm for the SAT problem based on surface computing and successfully validated the SAT instance $(w \wedge x \wedge y) \vee (w \wedge y \wedge z) \vee (x \wedge y) \vee (w \wedge y)$ through experimental implementation. The method can be simply described as: a group of single-stranded DNA molecules represent all possible solutions, a given computational problem is synthesized ("manufactured") and "fixed" on the surface, and the combination mixture of subsets is supplemented with "tags" in each N continuous cycle of DNA computation by the active functional group x, making them form double strands. After the "tagging" operation, an enzyme (for example, E. coli exonuclease I) is added, which can "destroy" the surface-bound oligonucleotides that exist in the form of unhybridized single strands ("destruction"). Then all hybrid chains are removed in the "untagging" operation.

In 2001, Wu [28] proposed a new improved algorithm for the SAT problem based on surface computing on the basis of Reference [17]. This algorithm has the following features:

1. The number of unique oligonucleotides that need to be synthesized is reduced from 2^n to $4n$, where n is the number of variables, reducing the cost.
2. Only three oligonucleotides need to be added, which greatly saves operation time.
3. The surface is reusable, because after the "untagging" operation, the surface is regenerated into a new surface.
4. The final result can be obtained without PCR amplification and then hybridization to an address array.

Plasmid DNA molecules are a type of circular superhelical structure. Reference [10] proposed to use plasmid DNA molecules to solve the SAT problem. In 2000, Faulhammer et al. [7] replaced DNA molecules with RNA molecules to provide an experimental computational model for the SAT problem, and discussed the RNA computational model of the chess problem.

In 2002, Braich et al. [3] applied the Sticker model to provide a DNA computational model for the SAT problem with 20 variables, and achieved success through biochemical experiments. Braich et al. proposed a DNA computing device and called it a DNA computer. On this DNA computer, they used a 3-SAT problem with 20 variables as an example for solving. After a brute force search of 1 million (2^{20}) possibilities, they found its unique answer .

The basic idea of this model is related to the paste model proposed by Roweis et al. [22]. The paste model includes two basic computational operations: separation and pasting operations based on sub-sequences (this study focuses solely

on the separation operation). Initially, the researchers filled glass modules with polyacrylamide gel and fixed oligonucleotide probes (single-stranded DNA) in the glass modules to achieve separation. The DNA strands carrying information move in the module through electrophoresis, and the DNA strands complementary to the fixed probe sequence are captured and retained in the module, while the non-complementary DNA strands pass through the module relatively freely. Subsequently, electrophoresis is performed again at a temperature higher than the melting temperature of the double-stranded body composed of the probe-DNA information chain, and the captured DNA chain can be released. During the separation process, covalent bonds are neither formed nor broken, and the DNA chain and glass module can be used for calculation multiple times.

The detailed algorithm steps are as follows [3]:

1. All variable truth values are represented using Lipton encoding. For each of the 20 variables, two different 15 bp "value sequences" $X_K^Z (k = 1 \ldots 20, \quad Z \in \{T, F\})$ are used to represent $x_k (k = 1 \ldots 20)$: one represents "true" (T, X_K^T), and the other represents "false", (F, X_K^F) . $\overline{X}_K^Z$ represents the Watson-Crick complementary sequence of X_K^Z .
2. Construction and detection of library chains. Each truth value assignment is represented by a 300 bp "library sequence". The library sequence is formed by the ordered connection of a value sequence of each variable. Single-stranded DNA molecules with library sequences are called "library chains". The set of all library chains and their complementary chains is called a "complete library". Two "half-libraries" are created first and then combined to form a complete library. The left half-library is used for x_1 to x_{10}, and the right half-library is used for x_{11} to x_{20} . After the construction of the library chain is completed, a "capture layer" is created by adding the corresponding acrylate-modified probe to the polyacrylamide gel to test the library chain. For each probe, about half of the chains in the half-library are captured, and about half of the chains pass through. The results show that for each variable, the number of half-library chains representing the truth value as "true" is roughly equal to the number of half-library chains representing the truth value as "false". Subsequently, PCR technology is further used to prove that the expected position of each variable in the library chain is correct.
3. Non-solution deletion, which specifically includes the following four operation steps:

Step one: Insert the library module into the hot chamber of the electrophoresis box, and insert the first clause module into the cold chamber of the electrophoresis box. In the hot chamber, the library chains in the library module are unchained with the acrylate-modified probe chains and migrate to the cold chamber where the first clause module is located. The library chains that satisfy the truth value assignment of the first clause are captured in the capture layer, while the library chains that encode the unsatisfied assignment pass through the capture layer and continue into the buffer library. For example, library chains with sequences X_3^F , X_{16}^F or X_{18}^T are

retained in the capture layer, while those with sequences X_3^T , X_{16}^T or X_{18}^F pass through the capture layer.

Step two: Replace the calculation module of the cold and hot chambers, insert the module in the cold chamber into the hot chamber, insert the next clause module into the cold chamber, and repeat the first step.

Step three: Repeat the above steps for the remaining 22 clauses. At the end of the calculation, the final clause module (the 24th clause module) will contain those library chains captured in all 24 clause modules, which are the truth value assignments that satisfy each clause of Φ and the truth value assignment of formula Φ.

Step four: Extract the solution chain from the last clause module, perform PCR amplification, and "read" the solution. Use primer group $\langle X_1^T, \overline{X}_K^T \rangle$, $\langle X_1^T, \overline{X}_K^F \rangle$, $\langle X_1^F, \overline{X}_K^T \rangle$, $\langle X_1^F, \overline{X}_K^F \rangle$ $(k = 2, 3 \ldots 20)$ to amplify each k value, and the corresponding gel electrophoresis shows that there is only one primer combination with the expected length of the band. According to the band information, the truth value of each variable is obtained. The truth value of each variable obtained in the experiment is the only solution to formula Φ.

6.3 DNA Computing Model of the Maximum Clique and Maximum Independent Set Problem of the Graph

Definition 6.2 Let G be a simple graph, and S be a non-empty vertex subset of G. If the induced subgraph $G[S]$ of S in G is a complete subgraph (complete null graph) of G, then S is a clique (independent set) of G. Let S be a clique of graph G, if for any clique (independent set) S' in graph G, $|S'| \leq |S|$, Then S is called a maximum clique (maximum independent set) of graph G. Obviously, if S is a maximum clique of graph G, then S is the maximum independent set of the complement $\overline{G}$ graph of G . Therefore, the problems of finding a maximum clique and a maximum independent set in a graph are equivalent.

The problems of maximum clique and maximum independent set in a graph belong to the difficult NP-complete problems. They not only have direct or indirect applications in the field of engineering technology, but also have good applications in mathematical theory itself, such as the famous Ramsey number problem [32] and so on. Therefore, it is of great significance to provide a fast and accurate algorithm for the maximum clique or maximum independent set of a graph. There are many research results in this area, such as conventional algorithms, neural network algorithms, and genetic algorithms [11, 29, 33]. However, these algorithms are designed with electronic computers as tools, so it is difficult to make substantial breakthroughs.

In 1997, Ouyang et al. [20] proposed a DNA solution to the problem of finding the maximum clique of a graph, imitating Adleman's method. The basic idea

is: first, a clique corresponds to a binary 0-1 sequence, and then a n-bit binary sequence corresponds to a DNA sequence with a length of $n > 30$. By applying the relationship between the graph and its complement, a DNA solution to the problem of finding the maximum clique of a graph is given. And a biochemical experiment of DNA calculation was carried out on a graph with 6 vertices, and it was successful. Below, we will briefly introduce Ouyang's work.

The algorithm steps to solve the maximum clique problem of a graph are as follows:

1. For a graph G with n vertices, each possible clique is represented by a n-bit binary number.
2. Construct the complement $\overline{G}$ graph of G .
3. Remove all numbers corresponding to vertices connected by an edge in the complement graph from the complete data pool.
4. Classify the remaining data pool and find the data with the most number of 1s. Each 1 in these data represents one vertex in the corresponding clique. Therefore, the data with the most number of 1s tells us the size of the maximum clique.

In addition to the computational model of Ouyang et al., in 2000, Head et al. [10] conducted experiments using their established plasmid DNA computational model. And compared this model with the work of Ouyang et al. in the paper. It pointed out that the application of plasmid DNA computation to solve the maximum clique and maximum independent set problems of graphs is superior to the double-stranded DNA computational model of Ouyang et al.

In 2002, Pan [21] proposed a DNA computational model for the maximum clique problem of a graph based on surface computation. The features of this model are: When the sample is fixed on a solid surface instead of in a solution, the sample handling is simpler and easier to automate; This model greatly reduces the loss of DNA strands in chemical operations, thereby reducing the error rate of DNA computation.

In 2012, Wang et al. [26] proposed a DNA self-assembly model for solving the maximum weighted independent set problem. This model mainly consists of two parts: a non-deterministic search system and an addition system. In the non-deterministic search system, each vertex is encoded as input, and all independent sets that meet the definition are identified according to the adjacency matrix of the graph. In the addition system, the total weight of each independent set is calculated by adding the weights of the vertices of each independent set, and finally the maximum weighted independent set is determined.

In 2013, Li [14] proposed a closed-loop DNA algorithm for the maximum weighted independent set problem based on the closed-loop DNA computational model and its biochemical experiments. In this algorithm, all independent sets are obtained first through appropriate coding and deletion experiments, and then the maximum weighted independent set is found through electrophoresis experiments and detection experiments.

In 2018, Yin et al. [5] were inspired by DNA origami nanotechnology and proposed to search for the maximum clique in an undirected graph with 6 vertices

and 11 edges. First, a unique type of scaffold is folded into a chain pool of n hairpin structures, or for convenience, this corresponds to a collection of 6 vertex cliques. Then, the scaffold is refolded with nails, and a series of selection processes are carried out according to the edges in the complementary graph. The scaffold of the correctly coded small clique is selected by gel electrophoresis. When this selection process ends, all cliques are found. By unfolding all the hairpins in the clique at the same time, and then performing gel electrophoresis, the scaffold with the largest number of hairpins in the clique is selected. Then, by unfolding the chain one by one to perform a series of unfolding of the hairpins outside the clique, it is determined whether each vertex in the given graph is in the maximum clique. When the decision process ends, the maximum clique in the given graph will be found. Our work shows that NP problems can be easily and reliably solved by using DNA origami through simple design of scaffolds, nails, and unfolding chains. This work further expands the application field of DNA origami technology.

6.4 DNA Computing Model for the 0-1 Programming Problem

0-1 programming is a special case of integer programming, where the variable x_i only takes the values 0 or 1, at which point x_i is called a 0-1 variable, or a binary variable. The application of 0-1 programming problems is very widespread. There are many algorithms for it, such as exhaustive method, implicit enumeration method, etc., but so far there is no good algorithm [34]. In 2018, Professor Yin Zhixiang's team [25] proposed a cyclic DNA model for the 0-1 programming problem based on DNA strand displacement on the previous basis. This model is based on circular DNA and contains multiple DNA recognition areas and small protection areas. It has higher recognition accuracy and more flexible structure than the normal DNA model. In Reference [25], the authors use DNA computation to solve the following special 0-1 programming problem,

$$\max(\min) z = c_1 x_1 + c_2 x_2 + \cdots + c_n x_n$$

$$\begin{cases} a_{11} x_1 + a_{12} x_2 + \cdots + a_{1n} x_n \leq (=, \geq) b_1 \\ a_{21} x_1 + a_{22} x_2 + \cdots + a_{2n} x_n \leq (=, \geq) b_2 \\ \vdots \\ a_{m1} x_1 + a_{m2} x_2 + \cdots + a_{mn} x_n \leq (=, \geq) b_m \end{cases}$$

Where x_j and a_{ij} are 0 or 1, b_i and c_j are non-negative integers.

The algorithmic steps for the 0-1 programming problem given in Reference [25] are as follows:

Step 1: Generate all possible combinations of variable values of 0 or 1 for the given problem;
Step 2: Use each constraint to eliminate infeasible solutions (retain feasible solutions);
Step 3: Generate the remaining solutions;
Step 4: Repeat steps 2 and 3 until all non-solutions are deleted to obtain all feasible solutions to the problem;
Step 5: Compare the objective function values corresponding to each feasible solution to obtain the optimal solution.

The corresponding biological experimental operation steps of the above algorithm are briefly described as follows:

Step 1: For a system of equations with n variables $x_1, x_2, \ldots, x_n$ and m equations, we first synthesize $4n$ short oligonucleotides. We divide it into 4 groups, the n oligonucleotides in the first group correspond to the variables $x_1, x_2, \ldots, x_n$, respectively; the n oligonucleotides in the second group correspond to the variables $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n$, respectively; the n oligonucleotides of the third group are the complementary chains corresponding to the first group, and are respectively denoted as $x_1', x_2', \ldots, x_n'$; the n oligonucleotides of the fourth group are the complementary chains corresponding to the second group, and are respectively represented as $\overline{x}_1', \overline{x}_2', \ldots, \overline{x}_n'$. In order to avoid their erroneous hybridization, the selected first two groups of oligonucleotides should have a large difference, at least 4 or more are different (note that here the oligonucleotide corresponding to x_i represents the variable x_i taking the value 1, the oligonucleotide corresponding to $\overline{x}_i$ represents the variable x_i taking the value 0).
Step 2: Connect the DNA single strands representing each variable x_i to form a circular DNA single strand. Then use the $2n$ oligonucleotides of the first two groups to construct 2^n DNA chains (all possible solutions of the variables).
Step 3: Put the constructed circular DNA chain into a test tube, and then put the constructed DNA single strand group into the step 2 test tube for DNA chain displacement reaction.
Step 4: By detecting the fluorescence amount after the reaction, the feasible solution satisfying the constraint equation is obtained.
Step 5: Repeat steps 2 and 3, we can obtain chains that satisfy all constraint equations.
Step 6: Calculate the objective function values corresponding to all feasible solutions and select the optimal solution.

Subsequently, Reference [24] proposed a model for the 0-1 integer programming problem based on the DNA chain displacement reaction network. This model uses the DNA chain displacement reaction as the basic principle and designs three chemical reaction modules, namely the weighted reaction module, the sum reaction module, and the threshold reaction module. These modules serve as the basic elements of the chemical reaction network and can be used to solve the 0-1 integer programming problem.

6.5 DNA Computing Model for the Graph Vertex Coloring Problem

Definition 6.3 Let $G(V, E)$ be a simple undirected graph, where V is the point set of $G(V, E)$, and E is the edge set of $G(V, E)$. The vertex coloring of a graph G refers to the assignment of a color to each vertex in G so that adjacent vertices are colored differently. In other words, it is a partition of the vertex set $V(G)$ of the graph G:

$$V(G) = V_1 \cup V_2 \cup \cdots \cup V_k, V_i \neq \emptyset, V_i \cap V_j = \emptyset, i = 1, 2 \ldots, k$$

The minimum number of colors required for a normal coloring of graph G is called the chromatic number, denoted as $X(G)$. A k-normal vertex coloring of graph G , also known as a k-vertex coloring of the graph, refers to coloring the graph G with $k(k \geq X(G))$ colors.

The graph coloring problem is a difficult combinatorial optimization problem with good application backgrounds, such as operation problems, scheduling problems, register allocation problems, etc., all have direct applications [2, 4] . Here we introduce it with an example of our research team's early research results [9].

For a graph with n vertices, each possible 3-vertex coloring scheme of the graph can be represented as an n-digit number string composed of 0, 1, and 2, where 0, 1, and 2 represent three colors respectively. The set of all possible coloring schemes of a graph with n vertices, transformed into an n-digit number string composed of 0, 1, and 2, is called a complete data pool. The theoretical algorithm steps are as follows:

Step 1: Encode the operation object and establish a complete data pool. Use the established complete data pool as the input data for DNA molecular computation. The encoding of each vertex consists of three parts, as shown in Fig. 6.2. The first segment P_i and the third segment P_{i+1} represent positions. The purpose is to further generate a complete data pool. The middle part V_i represents the encoding of the colors of each vertex. For each vertex, V_i has different encodings to represent different colors. And this part of the encoding uses oligonucleotide sequences with special enzyme cutting sites for solution separation.
Step 2: Search the data in the complete data pool and retain the data set that meets the coloring conditions.
Step 3: Provide the calculation results.

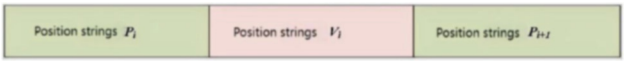


Fig. 6.2 Vertex encoding schematic

The following are the specific implementation steps:

Step 1: Establish a data pool, represent the data structure with double-stranded DNA, use i to represent the position, when i is odd, it is represented as $P_i V_i^m P_{i+1} (m = 0, 1, 2)$, when i is even, it is represented as $\overline{P_{i+1} V_i^m P_i}$. In order to effectively identify different colors of vertices, a restrictive sequence with special enzyme cutting sites must be added to distinguish the colors of vertices. Use parallel overlapping technology to establish a data pool.
Step 2: According to the definition of graph coloring, use restriction endonucleases to screen the DNA strings in the data pool. If 1 and 2 are two adjacent vertices in the graph, it is necessary to delete the DNA strings that represent vertices 1 and 2 of the same color. The deletion operation process is explained with the example of vertices 1 and 2 both being 0. First, divide the data in the data pool into two test tubes t_1 and t_2. In t_1, use EcoRI to cut the string containing V_1^0, and in t_2, use KpnI to cut the string containing V_2^0. Then merge the liquids in the two test tubes into test tube t, which does not contain data strings where vertices 1 and 2 are both 0. Repeat the above operation until the non-solution deletion is completed.
Step 3: Use polyacrylamide gel electrophoresis to identify the enzyme cutting products after the above operation is completed, and sequence the DNA chains. The coloring of each vertex is obtained according to the sequencing results.

In addition to using restriction endonucleases to delete non-solutions, a probe-magnetic bead separation technology has also been proposed to delete non-solution DNA computing models [31], and an experimental verification was completed using a 5-order graph as an example (Fig. 6.3).

The specific implementation steps are as follows:

Step 1: Encoding. For each vertex $i \in V(G)$ of graph G , let r_i, b_i, y_i represent the vertex i colored red, blue, and yellow respectively, and x_i , $x_i \in \{r_i, b_i, y_i\}$, are single-stranded oligonucleotide sequences containing l bases, and generate corresponding probe sequences based on these encodings. The probe library composed of these probe sequences is used to delete the non-solutions of graph G ;
Step 2: Synthesize the initial solution space. According to the marking order of graph G , synthesize DNA chains representing all possible solutions to represent all possible 3-vertex coloring schemes of graph G . This initial solution space should contain 3^n DNA sequences, and each sequence is $n \times l$ in length. At

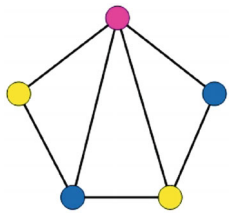


Fig. 6.3 A 5-order graph and one of its coloring schemes

the same time, synthesize probe sequences (biotin-labeled) for deleting non-solutions;
Step 3: Delete non-solutions. For those that do not meet the normal coloring conditions of graph G, hybridization with the single-stranded DNA in the initial solution space using a probe deletes the DNA strands that do not satisfy the normal 3-coloring of the graph, while retaining the remaining solutions;
Step 4: Repeat step 3, continue to exclude non-solutions, thereby retaining the DNA strands that represent solutions to the given problem;
Step 5: Use polymerase chain reaction to detect the solution.

In addition, in 1999, Jonoska et al. [12] theoretically described a general program to solve graph coloring problems and other computational problems through the self-assembly of complex molecular structures. They represent each k-degree vertex in the graph G as a k-arm DNA molecule subunit, these branched DNA molecules are called vertex building blocks. In addition to representing vertices, 2-arm structures are usually used for edge building blocks.

In 2009, Wu et al. [27] experimentally implemented a solution to the 3-coloring problem of a 6-order graph based on this. They encode vertices through branched DNA molecular structures and connect them to form a graph. Add specific restriction endonuclease sequences in the encoding to delete non-solutions.

In 2010, Lin et al. [15] proposed a non-deterministic graph vertex coloring algorithm, which ensures that under certain non-deterministic choices, a suitable vertex coloring scheme can be obtained. First, by separating adjacent information and coloring information, the input of the algorithm is obtained. Then, two reference tables are introduced: adjacency table and coloring table. The adjacency table is used to indicate the adjacency relationship of vertices in the graph, and the coloring table is used to assist in designing self-assembling components. Finally, based on the designed components, the non-deterministic algorithm is simulated through the self-assembly model. In the self-assembly process, possible coloring schemes are gradually constructed according to the characteristics and rules of the components, until a vertex coloring scheme that meets the conditions is found.

The algorithm is described as follows:
Non-Deterministic Algorithm (G, f)

(1) For each $V_i \in V(G)$ {
(2) Color for $V_i : f(i) \rightarrow \{r, b, y\}$
(3) Check all $(V_i, V_j) \in E(G)$ if exist $f(i) = f(j)$
(4) Break and return failure
(5) }
(6) If all $V_i \in V(G)$ are colored
(7) Return success and output $f(G)$
(8) Else return failure

Many researchers have studied the molecular computing model for solving the graph coloring problem from different perspectives, such as the computing model for solving the graph coloring problem using DNA quantum dots [13], and the self-

assembly computing model for the graph vertex coloring problem [30, 35] etc. We do not repeat them one by one in this book, and interested readers can refer to the corresponding literature for more details about these models.

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(5187), 1021–1024 (1994).
2. Berge, C.: Graphs and hypergraphs. North-Holland Pub. Co. (1973).
3. Braich, R. S., Chelyapov, N., Johnson, C., et al.: Solution of a 20-variable 3-sat problem on a dna computer. Science **296**(5567), 499–502 (2002).
4. Chaitin, G. J.: Register allocation & spilling via graph coloring. ACM Sigplan Notices **17**(6), 98–101 (1982).
5. Cui, J., Yin, Z., Yang, J., et al.: Searching for maximum clique by dna origami. In: 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 167–172. IEEE, Huangshan, China (2018).
6. Cukras, A. R., Faulhammer, D., Lipton, R. J., et al.: Chess games: a model for rna based computation. Biosystems **52**(1–3), 35–45 (1999).
7. Faulhammer, D., Cukras, A. R., Lipton, R. J., et al.: Molecular computation: RNA solutions to chess problems. Proceedings of the National Academy of Sciences **97**(4), 1385–1389 (2000).
8. Faulhammer, D., Lipton, R. J., Landweber, L. F.: Counting DNA: estimating the complexity of a test tube of DNA. Biosystems **52**(1–3), 193–196 (1999).
9. Xu, J., Gao, L.: DNA algorithm for vertex coloring of graphs. Journal of Electronics **31**(4), 494–497 (2003).
10. Head, T., Rozenberg, G., Bladergroen, R. S., et al.: Computing with DNA by operating on plasmids. Biosystems **57**(2), 87–93 (2000).
11. Holland, J. H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. MIT press (1992).
12. Jonoska, N., Karl, S. A., Saito, M.: Three dimensional DNA structures in computing. BioSystems **52**(1–3), 143–153 (1999).
13. Li, J., Song, Z., Zhang, C., et al.: A molecular computing model for graph coloring problem using DNA quantum dot. Journal of Computational and Theoretical Nanoscience **12**(7), 1272–1276 (2015).
14. Li, Q., Yin, Z., Chen, M.: Closed circle dna algorithm of maximum weighted independent set problem. In: Proceedings of The Eighth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA), pp. 113–121. Springer Berlin Heidelberg, Huainan, China (2013).
15. Lin, M., Xu, J., Zhang, D., et al.: 3D DNA self-assembly model for graph vertex coloring. Journal of Computational and Theoretical Nanoscience **7**(1), 246–253 (2010).
16. Lipton, R. J.: DNA solution of hard computational problems. Science **268**(5210), 542–545 (1995).
17. Liu, Q., Wang, L., Frutos, A. G., et al.: DNA computing on surfaces. Nature **403**(6766), 175–179 (2000).
18. Liu, W., Gao, L., Liu, X., et al.: Solving the 3-SAT problem based on DNA computing. Journal of Chemical Information and Computer Sciences **43**(6), 1872–1875 (2003).
19. Liu, X. J., Liu, W. B.: DNA computing model of weighted Hamiltonian path problem. Systems Engineering and Electronics **24**(6), 99–102 (2002).
20. Ouyang, Q., Kaplan, P. D., Liu, S., et al.: DNA solution of the maximal clique problem. Science **278**(5337), 446–449 (1997).

21. Pan, L., Xu, J.: A surface-based DNA algorithm for the maximal clique problem. Chinese Journal of Electronics **11**(4), 469–471 (2002).
22. Roweis, S. T., Winfree, E., Burgoyne, R., et al.: A sticker based model for DNA computation. In: DNA Based Computers, pp. 1–29. Citeseer, Princeton, USA (1996).
23. Sakamoto, K., Gouzu, H., Komiya, K., et al.: Molecular computation by DNA hairpin formation. Science **288**(5469), 1223–1226 (2000).
24. Tang, Z., Yin, Z., Wang, L., et al.: Solving 0–1 integer programming problem based on DNA strand displacement reaction network. ACS Synthetic Biology **10**(9), 2318–2330 (2021).
25. Tang, Z., Yin, Z., Yang, J., et al.: The circular DNA model of 0–1 programming problem based on DNA strand displacement. In: 2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), pp. 173–177. IEEE, Huangshan, China (2018).
26. Wang, Y., Bai, X., Wei, D., et al.: DNA self-assembly for maximum weighted independent set problem. Advanced Science Letters **17**(1), 21–26 (2012).
27. Wu, G., Jonoska, N., Seeman, N. C.: Construction of a DNA nano-object directly demonstrates computation. Biosystems **98**(2), 80–84 (2009).
28. Wu, H.: An improved surface-based method for DNA computation. Biosystems **59**(1), 1–5 (2001).
29. Xu, J., Bao, Z.: Neural networks and graph theory. Science in China Series F: Information Sciences **45**, 1–24 (2002).
30. Xu, J., Chen, C., Shi, X.: Graph computation using algorithmic self-assembly of DNA molecules. ACS Synthetic Biology **11**(7), 2456–2463 (2022).
31. Xu, J., Qiang, X., Fang, G., et al.: A DNA computer model for solving vertex coloring problem. Chinese Science Bulletin **51**, 2541–2549 (2006).
32. Xu, J., Wong, C. K.: Self-complementary graphs and Ramsey numbers Part I: the decomposition and construction of self-complementary graphs. Discrete Mathematics **223**(1–3), 309–326 (2000).
33. Bao, Z., Xu, J.: Neural networks and graph theory. Chinese Science (E edition) **31**(6), 533–555 (2001).
34. Xu, J., Yin, Z., Zhang, F.: DNA computation model of 0-1 programming problem. Journal of Electronics and Information Technology **25**(1), 62–66 (2003).
35. Zhang, X., Niu, Y., Cui, G., et al.: Application of DNA self-assembly on graph coloring problem. Journal of Computational and Theoretical Nanoscience **6**(5), 1067–1074 (2009).

Chapter 7
Non-enumerative DNA Computation Model for Graph Vertex Coloring

The previous chapter introduced the enumerative DNA computation model for solving NP-complete problems, but with the increase of problem size, the amount of DNA molecules in the generated initial solution space will inevitably show an "exponential explosion". Roughly estimated, for solving the 4-coloring problem of a graph of order 200, the solution space built upon the enumeration approach would contain 4^{200} potential solutions. Assuming that a DNA sequence representing a vertex is 50 bases in length, then the entire DNA molecule representing the feasible solution of this problem would be 10,000 bases long. This makes the mass of the DNA molecules in the solution space far exceed the mass of the Earth. Sakamoto et al. put forward a hairpin DNA computing model in the reference [1], indicating that for solving larger—scale SAT problems, it is essential to improve the encoding method. This improvement aims to reduce the number of DNA molecules needed, meaning that a non-enumerative DNA computing model ought to be established. The reference [2] first proposed a non-enumerative DNA computing model for solving graph vertex coloring problems. In this chapter, we will introduce this model in detail with permission from IEEE Transactions on NanoBioscience, Jin Xu, "An Unenumerative DNA Computing Model for Vertex Coloring Problem", 2011; all rights reserved.

7.1 Basic Idea of The Non-enumerative DNA Computing Model

In this DNA computing model, the encoding problem and the initial solution space problem are jointly considered. The encoding scheme and the construction of the initial solution space are optimized. By reducing the number of encodings representing colors according to the constructure of the given, the number of DNA molecules in the initial solution space is greatly reduced when constructing the initial solution space, thereby overcoming the problem of "exponential increase

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_7

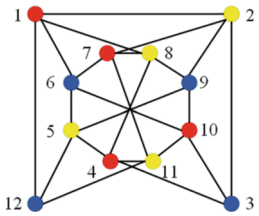


Fig. 7.1 Unique 3-colorable graph with 12 vertices [3]

in the solution space as the problem scale increases". First, during encoding, the number of oligonucleotide fragments of different colors representing each vertex is as small as possible; second, the adjacent vertices in the graph are also adjacent when marked. This computing model mainly uses PCR to implement non-solution deletion operations, and DNA sequencing technology verifies its final solution.

The algorithm steps of the non-enumerative DNA computing model are as follows:

Step 1: Determine the color set;
Step 2: Design and synthesize oligonucleotide fragments representing different vertices and different colors, and determine the probes synthesized for constructing the initial solution space;
Step 3: Generate the initial solution space;
Step 4: Delete non-solutions and retain the remaining solutions generated;
Step 5: Repeat step 4 until all non-solutions are deleted;
Step 6: Solution detection.

The specific biological implementation process of this model is explained in detail below using the 12-order graph in Fig. 7.1 as an example.

The graph G shown in Fig. 7.1 is a unique 3-color graph without triangles and is also a Hamilton graph. The vertex subsets {1, 4, 7, 10} are colored red, {2, 5, 8, 11} are colored yellow, and {3, 6, 9, 12} are colored blue.

7.2 Biological Implementation of The Non-enumerative DNA Computing Model

7.2.1 Biological Operation Steps

Corresponding to the above DNA computing model and algorithm steps, the specific biochemical operations can be simply described as follows:

Step 1: According to the given graph, establish a mapping from the vertex color set to the oligonucleotide sequence set, that is, encode the colors of the vertices of the graph to be solved according to certain constraints, and design corresponding probe sequences. These encoding sequences and probe sequences will be used to establish the initial solution space;

Step 2: Mix the DNA sequences of color set and probe, use hybridization reaction, PCR and other technologies to generate DNA stands representing all possible solutions of the given problem;
Step 3: Use PCR reaction to extract and amplify DNA stands representing normal coloring of the graph, thereby eliminating non-solutions. DNA stands representing normal coloring of the graph are separated and purified by agarose gel electrophoresis;
Step 4: Use the product recovered in the previous step as a feasible solution library, repeat step 3 until all non-solutions are eliminated, and the remaining DNA sequences are the solutions representing the problem;
Step 5: Sequence and analyze the sequence to obtain the coloring scheme. In the following content, we will give the specific biochemical experimental methods and experimental results corresponding to each step.

7.2.2 Case Analysis and Related Biochemical Experiments

(1) Encoding
For graph G in Fig. 7.1, when vertices 1 and 12 are colored red and blue, respectively, i.e., r_1 and b_{12}, the coloring schemes of the vertices adjacent to vertices 1 and 12 can be determined accordingly. For example: Since vertex 1 is colored red, the coloring schemes of vertices 2, 6, and 8 adjacent to vertex 1 are blue or yellow; and vertex 12 is colored blue, so the coloring schemes of vertices 5 and 11 adjacent to vertex 12 are red or yellow.

Let x_i represent the coloring of vertex i, here $x_i \in \{r_i, y_i, b_i\}$, $i = 1, 2, \ldots, 12$. Therefore, according to the above analysis, the coloring schemes of each vertex of graph G are as shown in Fig. 7.2.

All these possible k-colorings of graph G are denoted as $C_k(G)$. Here, $X_i, i = 1, 2, \ldots, 12$ represents the set of all possible coloring schemes for vertex i. According to the principle that adjacent vertices cannot be colored the same, it is obvious that $X_2 = \{y_2, b_2\}$, $X_3 = \{r_3, y_3, b_3\}$, $X_4 = \{r_4, y_4, b_4\}$, $X_5 = \{r_5, y_5\}$, $X_6 = \{y_6, b_6\}$, $X_7 = \{r_7, y_7, b_7\}$, $X_8 = \{y_8, b_8\}$, $X_9 = \{r_9, y_9, b_9\}$, $X_{10} = \{r_{10}, y_{10}, b_{10}\}$, $X_{11} = \{r_{11}, y_{11}\}$, and $C_k(G) = \{x_1x_2 \ldots x_{12}, x_i \in X_i, 1 \leq i \leq 12\}$.

1	2	3	4	5	6	7	8	9	10	11	12
r_1		r_3	r_4	r_5		r_7		r_9	r_{10}		
	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9	y_{10}	y_{11}	
	b_2	b_3	b_4		b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}

Fig. 7.2 When it is determined that $X_1 = \{r_1\}$, $X_12 = \{b_12\}$, the possible coloring schemes of each vertex of graph G [2]

As can be seen in Fig. 7.2, there is no need to enumerate all possible colorings for each vertex here, which reduces the required encoding. That is to say, only 27 oligonucleotide fragments need to be designed here to represent the possible colorings of different vertices. This not only lays the foundation for overcoming the problem of exponential explosion of the solution space, but also allows for shorter base sequences to encode vertices. This experimental scheme is mainly executed through PCR reactions, and the quality of primers in PCR directly affects the success or failure of PCR experiments. Therefore, in order to obtain highly specific encoding, the principles of primer design in biochemical experiments must be considered. The principles of primer design are as follows [4, 5].

First, the primer and the template sequence (the DNA sequence to be amplified) must be tightly complementary, secondly, the primer itself or the upstream and downstream primers should avoid forming stable dimers or hairpin structures, and thirdly, the primer must be specific to the template sequence, i.e., it cannot initiate DNA polymerization reactions at non-target sites on the template (i.e., mismatches).

In order to achieve these three basic principles in biological reactions, factors such as the inherent factors of the DNA sequence as a primer and specific experimental requirements should be considered, such as primer length, primer Tm value (melting temperature), primer GC content (composition), internal stability of the double strand formed by the primer and template, primer dimer, cross dimer between primers and hairpin structure, product length, etc.

In order to obtain a sufficient number of reliable encodings, the authors made trade-offs among the above biological conditions, and gave specific parameters for each condition in the encoding conditions, including sequence length, GC content, minimum substring length, and thermodynamic constraints, for program design. The specific methods have been introduced in Chap. 5 and will not be repeated here. The encoded DNA sequences satisfy the following constraints:

- Each designed encoding consists of 20 bases, with A, T, G, C randomly distributed;
- No encoding has more than 4 consecutive identical bases;
- The GC content of any encoding is between 40% and 60%;
- No two encodings have more than 6 consecutive identical base sequences;
- No encoding has a complementary sequence of 4 consecutive bases;
- The last 4 bases at the $3'$ or $5'$ end of any encoding cannot be the same as any 4 bases in other encodings.

Based on the above constraints, the authors used the principles of primer design and the biochemical experiment computer-aided software Primer Premier 5.0 to verify the oligonucleotide fragments constituting the primers and obtained suitable DNA encoding for this model. Based on the constraints described above, the principles of primer designing were utilized and the software Primer Premier 5.0 was used to validate the DNA sequences that make up the primers. Through this process, DNA sequences suitable for this model were obtained. Let $x_i, x \in$

Table 7.1 Represents the oligonucleotide sequences of possible colorings for each vertex

x_i	Sequence	x_i	Sequence
r_1	$CTGGTCCTCTCCTCTAATCC$	y_2	$TCACCTACTACCTTCCCAAA$
b_2	$CCATTCACACACCTTCACTC$	r_3	$TAACTCCACCATAACCACAA$
y_3	$TACCATCATTCCTATCACCC$	b_3	$AACTTCTACCCTCAACCTCA$
r_4	$TCTTGAGCACTGACCTGACA$	y_4	$TCAGTCGGCATCAACATAGT$
b_4	$TACTTTCCACTTCCTAACCC$	r_5	$TGTTCGCAATCTATTCTCAG$
y_5	$TACAGGCTCTTCAGAACGAT$	y_6	$CATCACATAGCACTCCATCG$
b_6	$CACAACCAGACCTGCTATCA$	r_7	$ATAGTTCCGAGTCTTAGGCA$
y_7	$TTACACGAGCGTCTTCTGAT$	b_7	$CAATGGCAACGATAACTTTC$
y_8	$TACATTCAAGGACGACAGGT$	b_8	$GAAGTCATCGTTGGGTAGTC$
r_9	$GATGATAAGCACGGAGTAGC$	y_9	$GTTACGGTTGTCTTTGCTGA$
b_9	$TAAAGTGTAGGGAGGGAAAC$	r_{10}	$TTGAACACAGGTATGCGATT$
y_{10}	$CCGTTATGAGCAGGTGTAAT$	b_{10}	$ACGACACGACGAGATAGGAT$
r_{11}	$TTAGTGAGAATGCCAGTTGC$	y_{11}	$CGTGATTGTTGGACTATTGG$
b_{12}	$CTTACCGCCTTACCAACTAC$		

The direction of all sequences are from 5′ to 3′

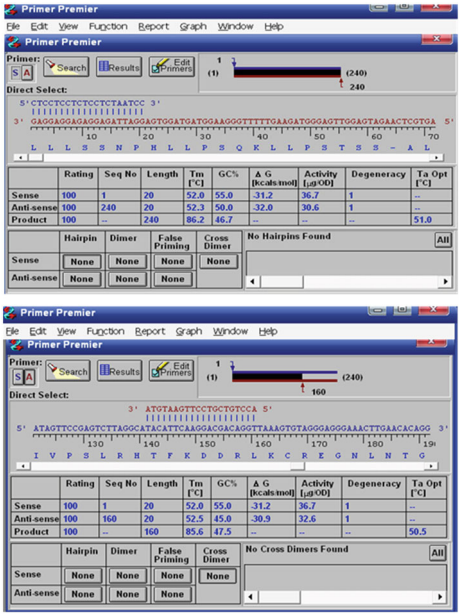


Fig. 7.3 Partial encoding verification results [2]

$\{r, b, y\}$, $i = 1, 2, \ldots, 12$, denote the DNA sequences corresponding to the possible colorings of each vertex in graph G, and its Watson-Crick complementary sequence is represented by $\overline{x_i}$. The specific DNA sequence corresponding to each x_i is shown in Table 7.1, and the verification results of these sequences are shown in Fig. 7.3.

The verification results of the biochemical experiment computer-aided software show that the Tm values of most primer pair sequences do not differ much, and there are basically no secondary structures that affect the normal progress of the reaction. The designed sequences can be used for experimental research of this model.

(2) Design of the Probe Library

Construct the initial non-enumerative solution space in accordance with the principle of normal coloring. This principle stipulates that adjacent vertices within the graph must be colored with different colors. Because vertex i and vertex $i+1$ are always adjacent, $i = 1, 2, \ldots, n-1 (n = 12)$, so the constructed initial solution space (denoted as T) should be: $T = \{x_1 x_2 \ldots x_{11} x_{12}\}$, where $x_i \in \{r_i, y_i, b_i\}$, $i = 1, 2, \ldots, 12$, and x_i and x_{i+1} in x are not both r , y or b. For example, when $i = 3$, there is no b_3 and b_4 appearing in a sequence in T, that is, adjacent vertices are not allowed to be dyed the same color.

Let the DNA sequence $x_i \in X_i$ representing the color of vertex i be divided into the first 10 bases as x_i^1 , and the last 10 as x_i^2, $i = 1, 2, \ldots, 12$. In order to establish the database T, take the last 10 DNA fragments x_i^2 in x_i and the first 10 DNA fragments x_{i+1}^1 in x_{i+1} to synthesize a 20 bp DNA sequence as a probe, marked with $\overline{x_i x_{i+1}}$. For example, for the sequence $y_2 = 5' - TCACCTACTACCTTCCCAAA - 3'$, $r_3 = 5' - TAACTCCACCATAACCACAA - 3'$, $b_4 = 5' - TACTTTCCACTTCCTA ACCC - 3'$, then there is $y_2 r_3 = 5' - CCTTCCCAAATAACTCCACC - 3'$, $r_3 b_4 = 5' - ATAACCACAATACTTTCCAC - 3'$, hence $\overline{y_2 r_3} = 5' - GGTGGAGTTATTTGGGAAGG - 3'$, $\overline{r_3 b_4} = 5' - GTGGAAAGTATTGTG GTTAT - 3'$. According to this method, 43 probes were constructed, as shown in Table 7.2.

(3) The Initial Solution Space Construction

The 27 DNA sequences representing colors are connected in the order of the vertex labeling in graph G under the action of T4 DNA ligase through the above 43 probes to form the DNA strands representing all possible solutions of the initial solution space. Since each DNA sequence contains 20 bases, each DNA strands in the initial solution space is 240 bp long. Figure 7.4 explains its experimental principle.

Based on the sequence of the calibrated vertices, it is not difficult to see that the vertices i and $i+1$, $i = 1, 2, \ldots, 11$, are always adjacent. Use T to represent the generated initial solution space, that is, the set $T = \{x_1 x_2 \ldots x_{11} x_{12}\}$, $x_i \in X_i$, $i = 1, 2, \ldots, 12$, and the vertices i and $i+1$, $i = 1, 2, \ldots, 11$, cannot be dyed the same color.

The specific steps to synthesize the initial solution space are as follows:

Step 1: $5'$ Phosphorylation. The sequence representing each vertex is dyed differently under the condition of T4 polynucleotide kinase, and incubated at 37 °C for 1h. After the reaction, the reaction product is marked as a phosphorylated product.

Table 7.2 43 probes and their sequences

$\overline{x_i x_{i+1}}$	Sequence	$\overline{x_i x_{i+1}}$	Sequence
$\overline{r_1 y_2}$	*TAGTAGGTGAGGATTAGAGG*	$\overline{r_1 b_2}$	*GTGTGAATGGGGATTAGAGG*
$\overline{y_2 r_3}$	*GGTGGAGTTATTTGGGAAGG*	$\overline{y_2 b_3}$	*GGTAGAAGTTTTTGGGAAGG*
$\overline{b_2 r_3}$	*GGTGGAGTTAGAGTGAAGGT*	$\overline{b_2 y_3}$	*AATGATGGTAGAGTGAAGGT*
$\overline{r_3 y_4}$	*TGCCGACTGATTGTGGTTAT*	$\overline{r_3 b_4}$	*GTGGAAAGTATTGTGGTTAT*
$\overline{y_3 r_4}$	*GTGCTCAAGAGGGTGATAGG*	$\overline{y_3 b_4}$	*GTGGAAAGTAGGGTGATAGG*
$\overline{b_3 r_4}$	*GTGCTCAAGATGAGGTTGAG*	$\overline{b_3 y_4}$	*TGCCGACTGATGAGGTTGAG*
$\overline{r_4 y_5}$	*AGAGCCTGTATGTCAGGTCA*	$\overline{y_4 r_5}$	*ATTGCGAACAACTATGTTGA*
$\overline{b_4 r_5}$	*ATTGCGAACAGGGTTAGGAA*	$\overline{b_4 y_5}$	*AGAGCCTGTAGGTTAGGAA*
$\overline{r_5 b_6}$	*TCTGGTTGTGCTGAGAATAG*	$\overline{r_5 y_6}$	*CTATGTGATGCTGAGAATAG*
$\overline{y_5 b_6}$	*TCTGGTTGTGATCGTTCTGA*	$\overline{b_6 r_7}$	*TCGGAACTATTGATAGCAGG*
$\overline{b_6 y_7}$	*GCTCGTGTAATGATAGCAGG*	$\overline{y_6 r_7}$	*TCGGAACTATCGATGGAGTG*
$\overline{y_6 b_7}$	*GTTGCCATTGCGATGGAGTG*	$\overline{r_7 y_8}$	*CTTGAATGTATGCCTAAGAC*
$\overline{r_7 b_8}$	*CGATGACTTCTGCCTAAGAC*	$\overline{b_7 y_8}$	*CTTGAATGTAGAAAGTTATC*
$\overline{y_7 b_8}$	*CGATGACTTCATCAGAAGAC*	$\overline{y_8 r_9}$	*GCTTATCATCACCTGTCGTC*
$\overline{y_8 b_9}$	*CTACACTTTAACCTGTCGTC*	$\overline{b_8 r_9}$	*GCTTATCATCGACTACCCAA*
$\overline{b_8 y_9}$	*CAACCGTAACGACTACCCAA*	$\overline{r_9 y_{10}}$	*CTCATAACGGGCTACTCCgT*
$\overline{r_9 b_{10}}$	*GTCGTGTCGTGCTACTCCGT*	$\overline{y_9 r_{10}}$	*CTGTGTTCAATCAGCAAAgA*
$\overline{y_9 b_{10}}$	*GTCGTGTCGTTCAGCAAAGA*	$\overline{b_9 r_{10}}$	*CTGTGTTCAAGTTTCCCTCC*
$\overline{b_9 y_{10}}$	*CTCATAACGGGTTTCCCTCC*	$\overline{r_{10} y_{11}}$	*AACAATCACGAATCGCATAC*
$\overline{y_{10} r_{11}}$	*TTCTCACTAAATTACACCTG*	$\overline{b_{10} r_{11}}$	*TTCTCACTAAATCCTATCTC*
$\overline{b_{10} y_{11}}$	*AACAATCACGATCCTATCTC*	$\overline{y_{11} b_{12}}$	*AGGCGGTAAGCCAATAGTCC*
$r_{11} b_{12}$	*AGGCGGTAAGGCAACTGGCA*		

The direction of all sequences are from 5′ to 3′

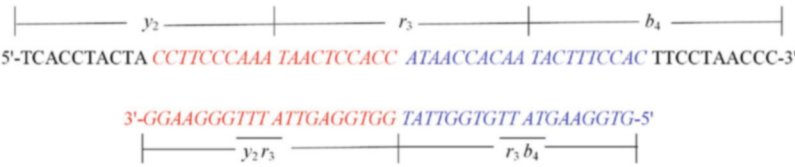


Fig. 7.4 Schematic diagram of the principle of synthesizing the initial solution space [2]

Step 2: Annealing. Take the reaction product from the previous step and mix it with all probes, heat to 94 °C, slowly cool to room temperature after 5min.
Step 3: Connection. Take 6 μl of the annealed product and add T4 DNA ligase, and leave it overnight at 16 °C.
Step 4: PCR amplification. After the above steps are completed, the obtained DNA sequence with a length of 240 bp represents the initial solution space of the 3-colorable problem of graph G (Fig. 7.5). To further purify and increase the content of each DNA molecule in the initial solution space, it is necessary to use the connection product as a template, and r_1 and $\overline{b_{12}}$ as primers to complete the PCR amplification operation of this step.

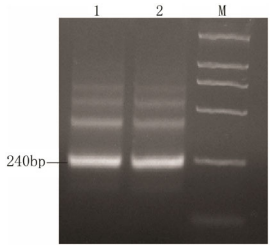


Fig. 7.5 Construction of the library. The M is the DNA marker DL2000 [2]

Step 5: Initial solution space detection. To detect the DNA sequence in the initial solution space constructed, dilute the recovered product from step 4 by 100 times, and use $< x_i, \overline{b_{12}} >$ and $< x_i, \overline{x_{i+1}} >$ as primers to check whether the library is complete. The PCR reaction system and reaction conditions are the same as above. The results show that there are appropriate DNA fragments produced at the corresponding positions, which proves that the initial solution space is basically complete.

It needs to be emphasized that, based on the sequence of the calibrated vertices, it is not difficult to see that the vertices i and $i + 1$, $i = 1, 2, \ldots, 11$, are always adjacent. Therefore, when constructing the initial solution space, do not use probes like $\overline{r_i r_{i+1}}$, $\overline{b_i b_{i+1}}$ and $\overline{y_i y_{i+1}}$ such probes. This also deletes the non-solutions where vertices i and $i + 1$, $i = 1, 2, \ldots, 11$ are dyed the same color, greatly reducing the constructed initial solution space. According to calculations, the generated initial solution space only contains 283 DNA sequences representing possible coloring schemes of graph G , which is far less than 3^{12}.

(4) Non-solutions Deletion

Let $C = 123\ldots(12)1$, if excluding C and edge $e = \{1, 6\}$, $e = \{1, 8\}$, $e = \{5, 12\}$, $E(G)$ is empty, then all DNA stands in the initial solution space constructed by the above steps represent the true solution of graph G. Otherwise, there must be non-solutions in the generated initial solution space. We use PCR, a biological operation, to delete non-solutions. For each remaining edge of $E(G)$, $e = ij$, $i, j = 1, 2, \ldots, 12$, $i \neq j$, three PCRs are required to delete the corresponding non-solutions.

In the first PCR, use $< r_1, \overline{x_i} >$, $< x_i, \overline{x_j} >$ and $< x_j, \overline{b_{12}} >$ as primer pairs for amplification. Here, obviously x_i and x_j cannot be r, b or y at the same time, so the DNA stand representing vertices i and j dyed the same color will not be amplified. For each edge, use the same operation method to delete all non-satisfying graphs G Solutions for normal 3-coloring. For example, with edge $e = \{6, 10\}$, the principle of removing non-solutions is shown in Fig. 7.6.

The first PCR operation: In this PCR reaction, primer pairs $< r_1, \overline{y_6} >$, $< r_1, \overline{b_6} >$, $< y_6, \overline{r_{10}} >$, $< y_6, \overline{b_{10}} >$, $< b_6, \overline{r_{10}} >$, $< b_6, \overline{y_{10}} >$, $< r_{10}, \overline{b_{12}} >$, $< y_{10}, \overline{b_{12}} >$ and $< b_{10}, \overline{b_{12}} >$ are used for amplification, respectively marked as ①–⑨. The DNA fragment sizes of products ① and ② are 120 bp, those of ③–⑥ are 100 bp, and those of ⑦–⑨ are 60 bp. The PCR results are shown in Fig. 7.7.

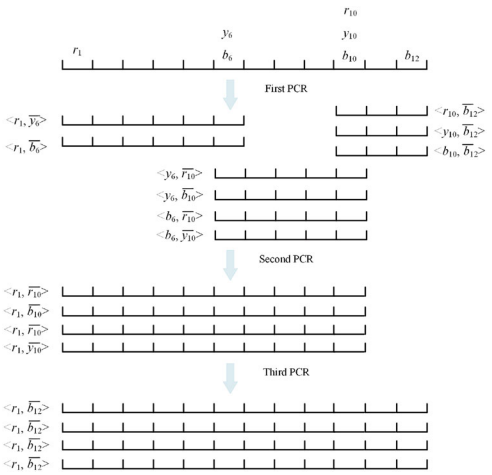


Fig. 7.6 Schematic diagram of non-solution deletion

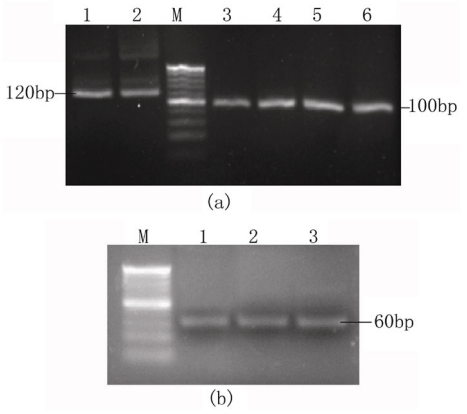


Fig. 7.7 Results of the first PCR experiment [2]

The products displayed in lanes 1 and 2 of Fig. 7.7a correspond to primer pairs $< r_1, \overline{y_6} >$, and $< r_1, \overline{b_6} >$. Lanes 3, 4, 5, and 6 correspond to primer pairs $< y_6, \overline{r_{10}} >$, $< y_6, \overline{b_{10}} >$, $< b_6, \overline{r_{10}} >$, and $< b_6, \overline{y_{10}} >$, respectively. In Fig. 7.7b, lanes 1, 2, and 3 correspond to primer pairs $< r_{10}, \overline{b_{12}} >$, $< y_{10}, \overline{b_{12}} >$ and $< b_{10}, \overline{b_{12}} >$. Through this PCR, a DNA sequence with a full length of 240 bp is decomposed into 3 DNA fragments of different sizes. Here, in the combination of primers with vertices 6 and 10, there are no primer pairs composed of the same color sequence, so in the PCR products, the non-solution DNA stands with vertices 6 and 10 dyed the same color are removed, and the DNA sequences representing the normal coloring of these two vertices are retained.

The second PCR operation: In this PCR, the primer pairs are $< r_1, \overline{r_{10}} >$, $< r_1, \overline{y_{10}} >$, and $< r_1, \overline{b_{10}} >$, the experimental results are shown in Fig. 7.8.

Lane 1 of Fig. 7.8 corresponds to primer pair $< r_1, \overline{r_{10}} >$, the template is a mixture of PCR products shown in lanes 1 and 3 of Fig. 7.7, represented as

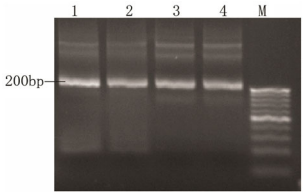


Fig. 7.8 Results of the second PCR [2]

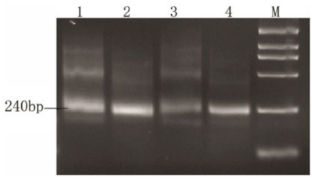


Fig. 7.9 Results of the third PCR [2]

①+③; lane 2 corresponds to primer pair $< r_1, \overline{b_{10}} >$, the template is ①+④; lane 3 corresponds to primer pair $< r_1, \overline{r_{10}} >$, the template is ②+⑤; the PCR product of lane 4 corresponds to primer pair $< r_1, \overline{y_{10}} >$, the template is ②+⑥.

The third PCR operation: The PCR products shown in Fig. 7.8 and the products shown in Fig. 7.7b are combined according to the color of vertex 10 as a new template. In this PCR, the only primer pair used is $< r_1, \overline{b_{12}} >$. The experimental results are shown in Fig. 7.9.

In Fig. 7.9, the lanes all use primer $< r_1, \overline{b_{12}} >$, the only difference is the combination of templates. The product of lane 1 corresponds to a mixture of the product in lane 1 of Fig. 7.8 and the product ⑦ from the first PCR; the product of lane 1 corresponds to a mixture of the product in lane 2 of Fig. 7.8 and the product ⑨ from the first PCR; the product of lane 3 corresponds to a mixture of the product in lane 3 of Fig. 7.8 and the product⑦ from the first PCR; the product of lane 4 corresponds to a mixture of the product in lane 4 of Fig. 7.8 and the product ⑧ from the first PCR.

For each remaining edge in the set $E(G)$, the same method is used to remove other non-solutions. When all edges are completed, the non-solutions of graph G in Fig. 7.1 are completely removed, and the remaining DNA sequences represent the normal coloring scheme of graph G.

(5) Solution Detection

In order to read the final solution that satisfies the normal 3-coloring of graph G, the product of the last round of PCR is connected to TaKaRa's pMD 19-T vector and transformed into E. coli DH5α. After PCR reaction identification and double enzyme cutting reaction identification, 25 single clones were selected for sequencing. The PCR reaction system and conditions are consistent with the above conditions. The results of PCR reaction identification and enzyme cutting reaction identification are shown in Figs. 7.10 and 7.11.

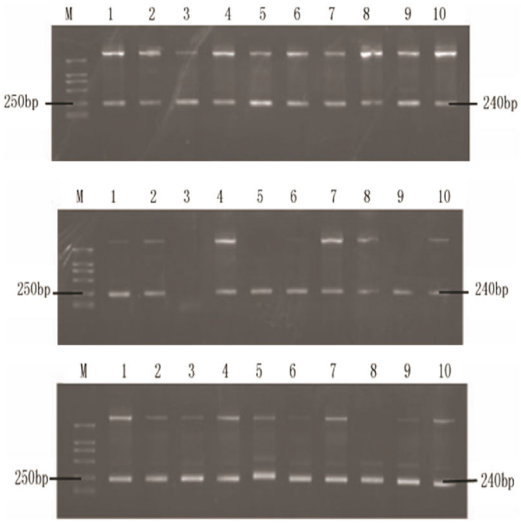


Fig. 7.10 PCR identification results [2]

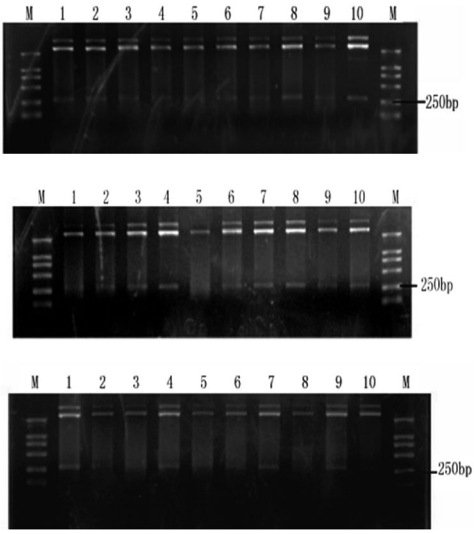


Fig. 7.11 Double enzyme cutting reaction identification results [2]

The sequencing results are shown in Fig. 7.12. The sequencing results were analyzed by Bioedit software, and as shown in Fig. 7.13, it can be known that there is only one solution for the graph.

The experimental results show that the coding scheme and the construction scheme of the initial solution space established by this model can greatly reduce the initial solution space, which is convenient for subsequent biochemical experiments; at the same time, the established coding parameters are reasonable, and the experiment has good repeatability.

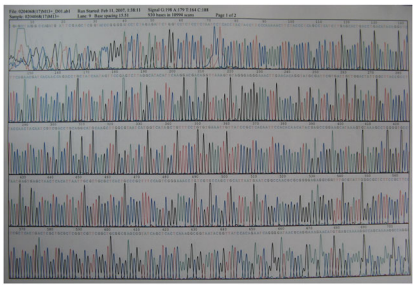


Fig. 7.12 Sequencing map

Fig. 7.13 Schematic diagram of the solution of graph G [2]

7.3 Analysis of Non-enumerative DNA Computing Model

As mentioned earlier, when vertex 1 and its associated vertex n are given a color in advance, namely red and blue, the number of oligonucleotide sequences representing possible colors of each vertex also decreases to $3n - d_1 - d_n - k$, where $d_1 = d_{12} = 2$, k is the sum of the degrees of vertex 1 and n minus 2. In past research, because the initial solution space used is an enumeration idea, the total number of DNA molecules synthesized representing various possible colors is $3n$.

For a graph with n vertices to perform $k(\geq 3)$ coloring, the possible number of colorings is k^n. When $k = 3$, it is 3^n. If a pair of adjacent vertices are determined to be colored, the number of searches is $3n - 2$. From Fig. 7.2 of this article, it can be seen that the color set of vertex i is X_i , $\{r_i, y_i, b_i\}$, the color set of vertex $i + 1$ is X_{i+1}, $\{r_{i+1}, y_{i+1}, b_{i+1}\}$, according to the principle of different coloring of adjacent vertices, when constructing the initial solution space, at most 2 elements are taken from the set C_i (or C_{i+1}), so that the initial solution space is less than 2^{n-2}. The reduction of the initial solution space eliminates some non-solutions during synthesis, thereby reducing the complexity and number of biochemical experiments, and is more economical. The more important point is that this idea can be used in other DNA computing models to reduce the initial solution space and reduce biological operation steps.

This model searches for feasible solutions through multiple rounds of PCR, each round of PCR includes 3 PCR reactions, and the total number of PCR reactions in this article is 24. If there are m edges remaining in the set $E(G)$, then the total number of PCR reactions $3m$.

Reference [2] is an experiment conducted on a Hamilton graph, and a Hamilton circle is selected to construct the initial solution space, thereby greatly reducing the solution space. The size of the established initial solution space is only 283, which is only 0.0532% of the initial solution space 3^{12} (531441) constructed by

the enumeration method, and most of the non-solutions are deleted at the same time when constructing the initial solution space. In fact, for non-Hamilton graphs, their operating principles are exactly the same, and the biggest difference is that the probes used to synthesize the initial solution space are different.

7.4 Other Non-enumerative DNA Computing Models

The non-enumerative DNA computing model provides a method to reduce the initial solution space of the problem to be solved, not only reducing the computational complexity of the algorithm, but also greatly reducing the complexity of biological operations. Subsequently, many researchers used this method to construct different DNA computing models.

In 2010, reference [6] built a "nano dial" molecular computing model based on circular DNA based on the non-enumerative idea, and solved the 3-coloring problem of the graph with 12 vertices using the backtracking deletion method. The main algorithm steps of this model are as follows:

Step 1: DNA Encoding and Generation of Solution Space
The first step of the calculation is to generate a DNA A molecular set is used to represent the complete data pool. In this step, the concept of non-enumeration is utilized. First, this concept is used to provide possible coloring schemes for each vertex, that is, the color set of each vertex (Table 7.3), and then the hybrid connection method is used to generate the initial solution space. This non-enumeration concept greatly reduces the number of DNA molecules in the feasible solution library, facilitating the execution of the subsequent backtracking algorithm.

Step 2: Backtracking Search in Graph G
In reference [6], an edge between two vertices that are not adjacent in numerical order is defined as a type 2 edge. The algorithm mainly uses the backtracking method to perform non-solution deletion operations on type 2 edges. In the type 2 edge (v_i, v_j), if $C_i \cap C_j \neq \emptyset$ (C represents the vertex color set), then the relationship between these two vertices is called backtracking. To complete this step, it is necessary to find the type 2 edges in graph G before searching for backtracking, and then determine whether the color sets of the two vertices satisfy $C_i \cap Cj_{/} = \emptyset$.

Table 7.3 Composition of the color set of vertices of a given 3-color graph

Vertex	1	2	3	4	5	6	7	8	9	10	11	12
Color set	r_1		r_3	r_4	r_5	r_6					r_{11}	
		y_2	y_3	y_4	y_5		y_7	y_8	y_9	y_{10}	y_{11}	
		b_2				b_6	b_7			b_{10}		b_{12}

Step 3: Non-solution Deletion by Backtracking
Backtracking deletion will make the two vertices of backtracking have different colors. To achieve this goal, reference [6] used several biological operations: Select (v_i, M, $t_i(r, y, b)$, M$t_i(r, y, b)$), Detect(H), Put into tube (B, M), Clear(M), and Mix(A, B, ..., Z). These operations can ensure that the DNA sequence representing the vertex color is determined in the final true solution. M denotes the solution space, v_i denotes the vertex of the graph, and t_i denotes a DNA sequence representing the vertex color (red (r), yellow (y), blue (b)). Mt_i denotes the set of DNA molecules containing only t_i. After selection, $H = 1$ if DNA products are present; otherwise $H = 0$. A, B, C, ..., Z denote the tube numbers.

Step 4: Solution Reading
Finally, all DNA stands representing the true solution are obtained. Through sequencing or DNA arrays, the true solution of the graph can be provided.

The problem of solution space explosion is the biggest bottleneck hindering the development of DNA computing. In response to this problem, the non-enumeration type DNA computing model established by optimizing the coding scheme and the construction of the initial solution space greatly reduces the number of DNA molecules required in the initial solution space. Reference [7] cited the non-enumerative DNA computing model in their paper and evaluated it as: "The researchers proposed an optimal method for generating graphical coloring solutions based on the optimization of biological operations using DNA computing technology. Although the experimental data set is small, it provides a new technique for solving problems, which can also be applied to finding solutions for graphs with a large number of nodes." The non-enumerative computing model not only theoretically reduces the solution space, but also has a relatively stable biological implementation method, which is also a basis for the parallel DNA computing model in the next chapter.

References

1. Sakamoto, K., Gouzu, H., Komiya, K. et al: Molecular computation by DNA hairpin formation. Science **288**(5469): 1223–12268 (2000).
2. Xu, J., Qiang, X., Yang, Y., et al.: An Unenumerative DNA Computing Model for Vertex Coloring Problem. IEEE transactions on Nanobioscience **10**(2): 94–98 (2011).
3. Harary, F., Hedetniemi, S.T., Robinson, R. W.: Uniquely colorable graphs. Journal of Combinatorial Theory **6**: 264–270 (1969).
4. Lin, W.: PCR Technology Operation and Application Guide. People's Military Medical Press, Beijing (1993).
5. Zheng Z.: Optimization Design of Oligonucleotides. Chemistry of Life **21**(3): 254–256 (2001).
6. Zhang, C., Yang, J., Xu, J. et al.: A "Nano-Dial" Molecular Computing Model Based on Circular DNA. Current Nanoscience **6**(3): 285–291 (2010).
7. Shukla, A., Bharti, V., Garg, M.L. A Greedy Technique Based Improved Approach to Solve Graph Colouring Problem. EAI Endorsed Transactions on Scalable Information Systems **8**(31): 4 (2021).

Chapter 8
Parallel Vertex Coloring DNA Computing Model

The non-enumerative DNA computing model presented in the preceding chapter has the remarkable ability to eliminate a vast number of non-solutions during the construction of the solution space. This effectively surmounts the issue of the exponential explosion of the solution space, thereby laying a solid foundation for further exploration into employing DNA molecules to address larger-scale and more complex problems. To further investigate the capability of DNA computing in solving larger-scale and more complex computational problems, the reference [1] proposes a parallel DNA computing model for the graph vertex coloring problem by comprehensively taking into account multiple perspectives, including the molecular level, algorithm level, and experimental level. Notably, the problem scale that this model can handle is the largest within the domain of DNA computing. The following section will focus on this remarkable work.

8.1 Model and Algorithm

The reference [1] puts forward several parallel basic ideas as follows: For larger-scale graphs, they are initially decomposed into several subgraphs. This subgraph partitioning approach not only enables the deletion of a greater number of non-solutions during the construction of the solution space but also facilitates the implementation of biological operations. Subsequently, each smaller subgraph is solved in parallel. The solution process encompasses a series of steps, including vertex sorting, the identification of bridge vertices (that is, a pair of adjacent vertices with the maximum degree within the subgraph), the determination of the color set for each vertex, DNA sequence encoding, the establishment of probes, the creation of the initial solution space, the elimination of non-solutions, and so on. Finally, the subgraphs are merged, and non-solutions are gradually deleted until the operation terminates upon the synthesis of the original graph. This model addresses the issue of the exponential explosion of the solution space through

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_8

optimizing subgraph partitioning, reducing vertex color sets, and sorting the vertices of subgraphs. Additionally, it designs a parallel PCR operation technology. By using this technology, non-solutions can be simultaneously deleted from multiple edges in the graph, significantly reducing the number of biological operations and substantially improving the computing efficiency. These parallel concepts render it feasible to solve larger-scale complex problems. The reference [1] performed experimental calculations on a graph with 61 vertices as shown in Fig. 5.4, and obtained all 8 solutions that satisfy the normal 3-coloring of the graph. It can be proved that after $v_1$and v_n are given coloring, the computing power of this model can reach $O(3^{59})$. The specific algorithm is shown in Algorithm 8.1:

Algorithm 8.1 Parallel DNA computing model algorithm

Input: Undirected connected graph G, color set C
Output: Normal coloring of graph G
subgraphs = divide_subgraphs(G) // Divide subsets for given graph G
Initialize color_set_dict = { }, probe_dict = { }, initial_solutions = { }
color_set_dict = encode_subgraph(subgraphs, C) // Determine the corresponding color set for the subgraph set
probe_dict = determine_probe(subgraphs) // Encode the subgraph set to determine the corresponding probe
for subgraph in subgraphs:// Traverse the subgraph set and generate the initial solution space corresponding to each subgraph
initial_solutions.append(generate_initial_solution(subgraph))
while not all_solutions_valid(subgraphs, initial_solutions): // Use PCR to delete non-solutions of each subgraph according to the graph path until all non-solutions are deleted
remove_non_solutions(subgraphs, initial_solutions)
merged = merge_and_remove(subgraphs, initial_solutions) // Merge subgraphs and delete non-solutions
while not all_solutions_valid(merged, initial_solutions): // Check if non-solutions have been completely removed, if not, continue to remove
remove_non_solutions(merged, initial_solutions)
return merged // Output solutions

8.1.1 Subgraph Partitioning and Determination of Bridge Vertices

The first step of the parallel DNA computing model is to perform subgraph partitioning. G_j, $V(G_j) \subset V(G)$, $E(G_j) \subset E(G)$, $j = 1, 2, \cdots, m-1$ are used to represent the first-level subgraphs. The steps for subgraph partitioning are as follows:

Step 1: Determine the size of the subgraph. Generally, the number of vertices in the partitioned subgraph is between 15 and 20 and each subgraph has as

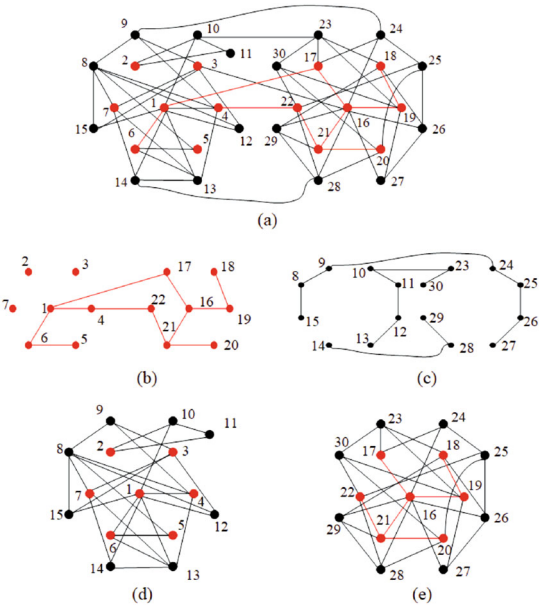


Fig. 8.1 An example of a first-level subgraph partition [1]. (**a**) Graph G. (**b**) Subgraph $G_{1'}$. (**c**) Subgraph $G_{2'}$. (**d**) Subgraph $G_3^{'}$. (**e**) Subgraph $G_4^{'}$

many edges as possible. This method of determining the first-level subgraph can eliminate most of the non-solutions during the processing of the subgraph. The more non-solutions are deleted in the first-level subgraph, the fewer non-solutions will naturally be in the synthesized second-level subgraph, which not only makes biochemical operations easier, but also prevents the complexity of the calculation from increasing too quickly with the gradual merging of the subgraphs.

For example, for the graph shown in Fig. 8.1, two different partition methods are given, where Fig. 8.1b and c are the results of the first partitioning method, Fig. 8.1d and e represent the second partitioning method. Although the size of the subgraphs obtained by the two partitioning methods is basically the same, the number of edges contained in each subgraph is different. Obviously, the second partitioning method is more likely to delete more non-solutions first in the subgraph.

Step 2: Determine the first-level subgraph and determine the two bridge vertices u_1, u_2 of this subgraph. These two bridge vertices must be adjacent and satisfy the condition that the sum of their degrees in the subgraph is the largest; if there is more than one pair of vertices that satisfy this condition, choose the pair with the most edges after sorting the vertices with these two points as the starting point and the end point, that is, satisfy $N_{edge}(u_1 = v_{i1}v_{i2}\cdots v_{in} = u_2)$ is the largest; if there is more than one pair of vertices that satisfy the above conditions, choose one of them.

Step 3: Determine the second subgraph G_2, this subgraph should include u_2. Find a new bridge point u_3 in subgraph G_2, the method is: the one with the

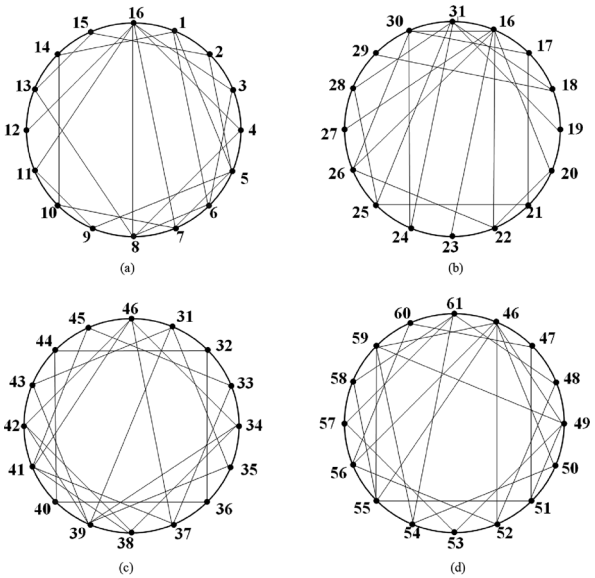


Fig. 8.2 The four subgraphs of the first-level subgraph partition of graph G [1]. (**a**) Subgraph G_1. (**b**) Subgraph G_2. (**c**) Subgraph G_3. (**d**) Subgraph G_4

largest degree in G_2 that is adjacent to u_2. The selection condition is the same as step 2.

Step 4: Repeat step 3 until the last subgraph is partitioned.

Figure 8.2 shows the 4 subgraphs divided from the graph G shown in Fig. 5.4, where vertices 1, 16, 31, 46, 61 are the 5 bridge vertices. It should be pointed out that the number of vertices in the last first-level subgraph G_m is generally different from the number of vertices in the previous subgraphs, and the bridge vertices of G_m should be u_1 (a bridge vertex of G_1) and u_{m-1} (a bridge vertex of G_{m-1}). The selection of bridge vertices is a key step in the model, because through the bridge vertices, the first-level subgraphs can be merged into second-level subgraphs using PCR technology. And the determination of the subgraph bridge vertices needs to meet the following conditions: in a subgraph, the sum of the degrees of the two bridge vertices should be as large as possible. The larger the sum of the degrees of the two bridge vertices, the more vertices adjacent to these two vertices, which will reduce the number of colors in the color set of the corresponding vertices, thereby making the initial solution space smaller.

8.1.2 Subgraph Vertex Sorting and Determination of Color Set of Each Vertex in Subgraph

8.1.2.1 Subgraph Vertex Sorting

After the subgraph partitioning is completed, the bridge vertices of each subgraph are also determined. Subgraph vertex sorting is to find a vertex sorting in this subgraph with the two bridge vertices as the starting point and the end point, so that the number of edges between adjacent vertices in this sorting is as many as possible. Take subgraph G_1 as an example, if the vertex set of subgraph G_1 is $V(G_1)$= $\{v_1, v_2, \cdots, vt\}$, let v_1, v_t be its two bridge vertices, without loss of generality, suppose the vertex sequence of the sorting is $v_1 = v_{i1}v_{i2}v_{i3}\cdots v_{it} = v_t$, it needs to satisfy

$$max|\{(v_{ij}v_{i(j+1)})\}|, v_{ij}, v_{i(j+1)} \in V(G_1); j = 1, 2, \cdots, t-1 \tag{8.1}$$

The concept of root path set is introduced before giving the vertex sorting algorithm.

Definition 8.1 Assume v is a vertex of graph G. The set of all paths in G starting from vertex v is called v-root path set, or simply v-path set, denoted as $P(v)$.

It is not difficult to find a vertex's path set, such as firstly taking the neighborhood $N(v)$, then finding the neighborhood for each vertex in $N(v)$, denoted as $N^2(v)$, $N^2(v)$ does not contain element v, and so on, resulting in $N^3(v)$, $N^4(v)$ and so on, which will terminate within $|V(G)| - 1$ steps. This method is actually the so-called pruning algorithm, an NP-algorithm. Since the number of vertices in the first-level subgraph determined in the model is within 20, it is easy to implement.

Next, the vertex sorting algorithm for the first-level subgraph will be introduced. Suppose G_1 is a first-level subgraph, v_1, v_t are two bridge vertices of G_1. The steps of the vertex sorting algorithm for G_1 are as follows:

Step 1: Find the root path sets $P(v_1)$ and$P(v_t)$ of the two bridge vertices v_1, v_t. If there is a Hamilton path of subgraph G_1 in $P(v_1)$, select this path. Otherwise, go to step 2;
Step 2: If $P(v_1) \cup P(v_t) = V(G_1)$, select two paths P_1 and P_t from $P(v_1)$ and $P(v_t)$ respectively. These two paths should meet the following three conditions: ① $P_1 = v_1v_{i2}v_{i3}\cdots v_{ir}$, $P_t = v_{is}v_{is+1}\cdots v_t$; ② $max(|E(P_1)| + |E(P_t)|)$; ③ $V(P_1) \cap V(P_t) = \emptyset$.
Step 3: If $V' = V(G_1) - P(v_1) - P(v_t) \neq \emptyset$, find the longest path from the derived subgraph $G[V']$, denoted as $P' = u_1u_2\cdots u_m$. If $V'' = V(G_1) - V(P_1) - V(P_t) - V' = \emptyset$, then the vertex sorting of subgraph G_1 is: $v_1v_{i2}v_{i3}\cdots v_{ir}u_1u_2\cdots u_mv_{is}v_{is+1}\cdots v_t$. If $V'' = V(G_1) - P(v_1) - P(v_t) - V' = u_{i1}, u_{i2}, \cdots, u_{iq} \neq \emptyset$, then the vertex sorting of subgraph G_1 is: $v_1v_{i2}v_{i3}\cdots v_{ir}u_1u_2\cdots u_mu_{i1}u_{i2}\cdots u_{iq}v_{is}v_{is+1}\cdots v_t$.
Take the subgraph G_3' shown in Fig. 8.1d as an example, its two bridge vertices are 1 and 13, you can find the path 1-10-11-2 from the root path set of bridge

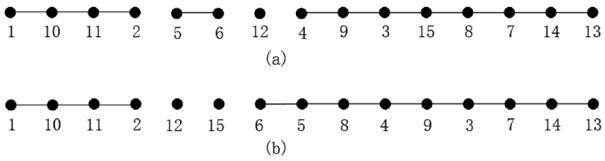


Fig. 8.3 Subgraph G_3' vertex sorting and corresponding set of 11 edges [1]

point 1; find the path 13-14-7-8-15-3-9-4 from the root path set of bridge point 13, then get $V' = \{5, 6, 12\}$, then find the longest path 5–6 from the derived subgraph $G[V']$, then get $V'' = \{12\}$. Therefore, the vertex sorting obtained is shown in Fig. 8.3a. Figure 8.3b shows another sorting method, and this sorting method is better than the sorting shown in Fig. 8.3a. As can be seen, the vertex sorting in the first subgraph in Fig. 8.2a is $1, 2, \cdots, 16$, this sorting is the best, each pair of adjacent vertices $i, i+1$ are adjacent, $i = 1, 2, \cdots, 15$.

Step 4: Vertex renumbering. After the vertices of the subgraph are sorted according to the above steps, the vertices of the graph need to be renumbered. Assume the vertex sorting of the subgraph before renumbering is $v_1 v_2, \cdots, v_t$, the sequence after the vertices are rearranged is $v_{\sigma(1)}, v_{\sigma(2)}, \cdots, v_{\sigma(t)}$, obviously, it is obtained by the following permutation of the vertex subscript:

$$\sigma = \begin{pmatrix} 1 & 2 & \cdots & t \\ \sigma(1) & \sigma(2) & \cdots & \sigma(t) \end{pmatrix}$$

For example, after the vertex sorting of the above subgraph G_3', the mapping of vertex renumbering is:

$$\sigma = \begin{pmatrix} 1\,2 & 3 & 4\,5\,6\,7 & 8\,9\,10\,11\,12\,13\,14\,15 \\ 1\,10\,11\,2\,5\,6\,12 & 4\,9\,3 & 15\,8 & 7\quad 14\,13 \end{pmatrix}$$

Without loss of generality, after the vertices of the sorted subgraph are renumbered, the principle of renumbering is: let the vertex $\sigma(1)$ be labeled as i, where $i = 1, 2, \cdots, t$.

8.1.2.2 Determination of Vertex Color Set

Suppose the color set $C_3(G) = \{r, b, y\}$, r_i, y_i, b_i represents the vertices v_i in the graph are colored red, yellow, and blue respectively. Since the bridge vertices of the subgraph are adjacent, without loss of generality, for the two bridge vertices v_1, v_t of the first level subgraph G_1, let vertex v_1 be colored red, denoted as r_1, and v_t be colored blue, denoted as b_t (Vertex v_t can also be colored yellow, just need to calculate all solutions in the case of blue, and then do a color permutation to get

Table 8.1 Composition of the color set of vertices of a given 3-color graph

Subgraph G_1		Subgraph G_2		Subgraph G_3		Subgraph G_4	
v_i	$C(v_i)$	v_i	$C(v_i)$	v_i	$C(v_i)$	v_i	$C(v_i)$
v_1	$\{r_1\}$	v_{16}	$\{b_{16}\}$	v_{31}	$\{r_{31}\}$	v_{46}	$\{b_{46}\}$
v_2	$\{y_2, b_2\}$	v_{17}	$\{r_{17}, y_{17}\}$	v_{32}	$\{y_{32}, b_{32}\}$	v_{47}	$\{r_{47}, y_{47}\}$
v_3	$\{r_3, y_3\}$	v_{18}	$\{y_{18}, b_{18}\}$	v_{33}	$\{r_{33}, y_{33}, b_{33}\}$	v_{48}	$\{r_{48}, b_{48}\}$
v_4	$\{r_4, y_4\}$	v_{19}	$\{y_{19}, b_{19}\}$	v_{34}	$\{r_{34}, y_{34}\}$	v_{49}	$\{r_{49}, y_{49}\}$
v_5	$\{y_5, b_5\}$	v_{20}	$\{r_{20}, y_{20}\}$	v_{35}	$\{y_{35}, b_{35}\}$	v_{50}	$\{r_{50}, y_{50}\}$
v_6	$\{y_6, b_6\}$	v_{21}	$\{r_{21}, y_{21}, b_{21}\}$	v_{36}	$\{r_{36}, y_{36}, b_{36}\}$	v_{51}	$\{r_{51}, y_{51}, b_{51}\}$
v_7	$\{r_7, y_7\}$	v_{22}	$\{r_{22}, y_{22}\}$	v_{37}	$\{r_{37}, y_{37}\}$	v_{52}	$\{r_{52}, y_{52}\}$
v_8	$\{r_8, y_8\}$	v_{23}	$\{r_{23}, y_{23}\}$	v_{38}	$\{r_{38}, y_{38}, b_{38}\}$	v_{53}	$\{r_{53}, y_{53}, b_{53}\}$
v_9	$\{r_9, y_9, b_9\}$	v_{24}	$\{y_{24}, b_{24}\}$	v_{39}	$\{y_{39}, b_{39}\}$	v_{54}	$\{r_{54}, b_{54}\}$
v_{10}	$\{r_{10}, y_{10}, b_{10}\}$	v_{25}	$\{y_{25}, b_{25}\}$	v_{40}	$\{r_{40}, y_{40}, b_{40}\}$	v_{55}	$\{r_{55}, y_{55}\}$
v_{11}	$\{r_{11}, y_{11}\}$	v_{26}	$\{r_{26}, y_{26}\}$	v_{41}	$\{r_{41}, y_{41}\}$	v_{56}	$\{r_{56}, y_{56}\}$
v_{12}	$\{r_{12}, y_{12}\}$	v_{27}	$\{r_{27}, y_{27}\}$	v_{42}	$\{r_{42}, y_{42}\}$	v_{57}	$\{r_{57}, b_{57}\}$
v_{13}	$\{r_{13}, y_{13}, b_{13}\}$	v_{28}	$\{y_{28}, b_{28}\}$	v_{43}	$\{y_{43}, b_{43}\}$	v_{58}	$\{r_{58}, b_{58}\}$
v_{14}	$\{y_{14}, b_{14}\}$	v_{29}	$\{r_{29}, y_{29}, b_{29}\}$	v_{44}	$\{r_{44}, y_{44}, b_{44}\}$	v_{59}	$\{r_{59}, y_{59}\}$
v_{15}	$\{r_{15}, y_{15}\}$	v_{30}	$\{y_{30}, b_{30}\}$	v_{45}	$\{r_{45}, y_{45}\}$	v_{60}	$\{r_{60}, b_{60}\}$
v_{16}	$\{b_{16}\}$	v_{31}	$\{r_{31}\}$	v_{46}	$\{b_{46}\}$	v_{61}	$\{y_{61}\}$

the solution of v_t colored yellow, that is, in the case of red unchanged, swap yellow and blue to get another coloring. The other subgraphs are treated in the same way.). Besides vertex v_t, each vertex in the neighborhood $N(v_1)$ of vertex v_1 is colored blue or yellow; similarly, except for vertex v_1, each vertex in the neighborhood $N(v_t)$ of vertex v_t is colored red or yellow. The color sets of the 4 first-level subgraphs shown in Fig. 8.2 are as shown in Table 8.1, where $C(1) = \{r_1\}$, $C(16) = \{b_{16}\}$, $C(31) = \{r_{31}\}$, $C(46) = \{b_{46}\}$, $C(61) = \{y_{61}\}$.

8.1.3 Encoding of DNA Sequences

Theorem 8.1 *Let the DNA sequence representing the possible colors of vertex vi be marked as* x_i, $x \in \{r, b, y\}$, $i = 1, 2, \cdots, n$. *Its Watson-Crick complementary sequence is represented by* $\overline{x_i}$. *According to the possible color set of each vertex, it can be determined that for any arbitrary graph, the number of DNA sequences that need to be encoded is:*

$$N_{DNA} = kn - d_1 - d_m - 2(k-1) \tag{8.2}$$

The encoding used in parallel DNA computing not only considers the specific hybridization reaction, but also considers the biological constraints of using PCR

technology, and also considers the factors of large scale. Comprehensive design, encoding according to the following constraints:

- All encodings do not have more than 4 consecutive A, T, C or G;
- The GC content of any encoding is between 40 and 60%;
- No two encodings have more than 8 consecutive bases that are the same;
- Any coding sequence does not have a complementary sequence of four consecutive bases;
- Any coding sequence's $3'$ or $5'$ end's five consecutive bases cannot be the same as any five bases in other codings;
- The absolute value of the chemical free energy change ($|\Delta G|$) generally does not exceed 6.0 kcal/mol when the DNA sequence used as a primer forms a dimer itself;
- The $|\Delta G|$ of the dimer formed between the DNA sequences used as primers generally does not exceed 9.0 kcal/mol, while the $3'$ end of the dimer formed between the primers $|\Delta G|$ cannot exceed 6.0 kcal/mol.

8.1.4 Determine the Calculation Probe According to the Probe Diagram

Probes are only established between two adjacent vertices determined in Sect. 8.1.2, that is, only between vertices v_i and v_{i+1}, the probe between them is denoted as $\overline{x_i x_{i+1}}$. If v_i and v_{i+1} are adjacent in G, then x_i and x_{i+1} cannot take the same color; if v_i and v_{i+1} are not adjacent in G , then x_i and x_{i+1} can take the same color. Here C_i represents the color set of vertex v_i, $i = 1, 2, \cdots, n$. A graph G's probe diagram, denoted as $B(G)$, is simply denoted as B, defined as:

$$V(B) = \bigcup_{i=1}^{n} C(i), \quad E(B) = \bigcup_{i=1}^{n} \{\overline{xz};\ \overline{xz}\ is\ a\ probe,\ x \in C(i),\ z \in C(i+1)\} \tag{8.3}$$

The specific steps to determine the calculation probe are as follows:

Step 1: Provide the probe diagram for each primary subgraph.
Step 2: Provide the probe set according to the vertices and edges of the probe diagram. For example, in the subgraph G_1 given in Fig. 8.3, the color set of vertex 7 is $\{r_7, y_7\}$, the color set of vertex 8 is $\{r_8, y_8\}$, according to the probe diagram of G_1 (Fig. 8.4a), the probes $\overline{r_7 y_8}$ can be obtained.
Step 3: Determine the DNA sequence representing each probe. For the probe $\overline{x_i x_{i+1}}$, it is composed of the complement of the sequence composed of the second half of the DNA sequence representing x_i and the first half of x_{i+1}. Such as y_7=$5' - AATACGCACTCATCACATCG - 3'$, r_8=$5' - GACCTTACCGTTTAGAGTCG - 3'$, then there is $y_7 r_8$=$5' -$

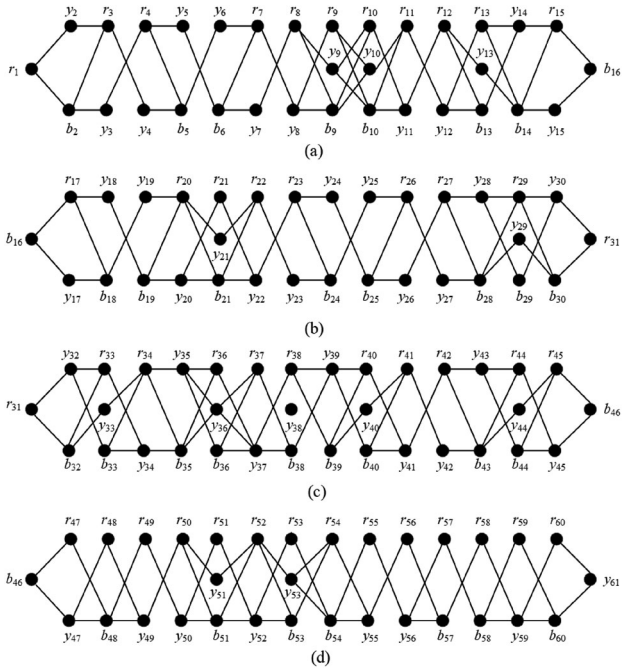


Fig. 8.4 Probe diagrams corresponding to 4 subgraphs. (**a**) B_1. (**b**) B_2. (**c**) B_3. (**d**) B_4

$CATCACATCGGACCTTACCG-3', \overline{y_7 r_8} = 5'-CGGTAAGG\ TCCGA\ TGTGATG-3'$.

Figure 8.4 shows the probe diagrams of the four primary subgraphs shown in Fig. 8.2, denoted as B_1, B_2, B_3 and B_4. From each probe diagram, the number and distribution of probes in the corresponding primary subgraph can be clearly seen.

8.1.5 Initial Solution Space Construction

The synthesis method of the initial solution space adopts the method proposed by Adleman [2] in 1994, but the synthesized initial solution space is no longer an enumeration of all possible solutions, but the remaining possible solutions after excluding a large number of non-solutions under the action of the previous steps. Here $S(G_j)$ represents the initial solution space of the primary subgraph G_j, that is, the set $S(G_j) = x_1 x_2 \cdots x_t$, where $x \in \{r, b, y\}$, $j = 1, 2, \cdots, m$. If each x_i is l bases long, and each subgraph has t vertices, then the generated DNA sequence representing the possible solution contains $t \times l$ bases.

It should be noted that when synthesizing the initial solution space of each primary subgraph, if there is an edge between vertices v_i and v_{i+1}, then the probe

between these two vertices does not include the situation where the two vertices have the same color, so when constructing the initial solution space, some non-solutions with the same color on adjacent vertices can be deleted. The larger the $N_{edge}(G_j)$ of the primary subgraph G_j, the more non-solutions are deleted when constructing the initial solution space.

8.1.6 Non-solution Deletion

Let $E_j^1 = \{(v_i v_{i+1}) \in E(G_j), \vee(v_i v_t) \in E(G_j); i = 1, 2, \cdots, t-1\}$, $E_j^2 = \{(v_1 v_i) \in E(G_j), i = 2, 3, \cdots, t\}$, $E_j^3 = E(G_j) - E|_j^1 - E_j^2$, $j = 1, 2, \cdots, m$. The removal of non-solutions is the deletion of the DNA sequences that color the two vertices associated with each edge in E_j^3 from the initial solution space $S(G_j)$. The process of removing non-solutions is mainly achieved through PCR technology. In the specific process of removing non-solutions, the edges in E_j^3 are divided into cases and non-solutions are removed separately.

(1) Forward Edge

Definition 8.2 Let G_j be a subgraph with a vertex subset $V(G_j) = v_1, v_2, \cdots, v_t$, and $v_1 v_2 \cdots v_t$ is its vertex sequence.$\forall e = v_i v_k \in E(G_j)$, if $1 \leq i < k \leq t$, then $e = v_i v_k$ is a forward edge; otherwise, it is a reverse edge. If all the edges in the path are forward edges, then the path is called a forward path, otherwise it is a reverse path.

The forward path $P = v_i v_j v_k v_l, 1 \leq i \leq j \leq k \leq l \leq t$ will be used as an example to introduce the principle of removing non-solutions in the case of forward edges. In this case, several PCRs need to be processed in parallel to complete the process of removing non-solutions. The path P contains three forward edges: $v_i v_j$, $v_j v_k$, $v_k v_l$, in the first PCR, $< x_1, \overline{x_i} >$, $< x_i, \overline{x_j} >$, $< x_j, \overline{x_k} >$, $< x_k, \overline{x_l} >$ and $< x_l, \overline{x_t} >$ are used as primer pairs for amplification. Except for the first pair and the last pair of primers, the two vertices in the other primer pairs cannot be colored the same. In this way, the DNA sequence is divided into 5 fragments, and each fragment's two endpoints represent a normal coloring for the corresponding vertices. Then they are merged step by step into $t \times l$ base DNA sequences. In this case, the non-solutions corresponding to multiple edges can be deleted in parallel.

(2) Single Edge

After removing non-solutions from E_j^3 according to the first case, some single edges may remain. The operation of removing non-solutions varies depending on whether the coloring of the two vertices associated with this edge is determined. After completing the removal of non-solutions from the edges in E_j^3 of each subgraph, the DNA sequences representing the normal coloring of each subgraph G_j can be obtained.

8.1.7 Subgraph Merging and Non-solution Deletion

After obtaining the DNA stands representing the solutions of each first-level subgraph, the subgraphs are merged step by step, and after each merge, the non-solution removal step for this level of subgraph is implemented, mainly including the following three steps:

Step 1: Merge the first-level subgraphs G_j and G_{j+1}, $j = 1, 2, \cdots, m$. Using the DNA sequence of the bridge point, the DNA sequences representing the normal coloring of these two first-level subgraphs after removing non-solutions are spliced together. If $|V(G_j)| = t_1$, $|V(G_{j+1})| = t_2$, then a DNA sequence of length$(t_1 + t_2 - 1) \times l$ bases is formed. These DNA sequences will serve as all possible solutions for the second-level derived subgraph $G[V(G_j) \cup V(G_{j+1})]$ of the graph G.
Step 2: According to the method of removing non-solutions described in Sect. 8.1.6, remove non-solutions from each second-level derived subgraph $G[V(G_j) \cup V(G_{j+1})]$. More precisely, remove the non-solutions corresponding to each edge in the edge set$E_{j,j+1} = \{(uv) \in E(G); u \in V(G_j), v \in V(G_j+1)\}$, and thus obtain the DNA sequences representing the normal coloring of the second-level derived subgraph $G[V(G_j) \cup V(G_{j+1})]$.
Step 3: Repeat steps 1 and 2 for the second-level or higher-level derived subgraphs until they are merged into the graph G, and finally obtain the DNA sequences representing the normal coloring of the graph G.

8.1.8 Solution Detection

The final DNA sequences representing the normal coloring of the graph can be easily obtained through sequencing.

8.2 Specific Example

8.2.1 Subgraph Partitioning and Color Set Determination

The method of subgraph partitioning and vertex sorting in Sect. 8.1.1 gives 4 subgraphs of graph G as shown in Fig. 8.2. According to the method in Sect. 8.1.1, the color sets of the 4 subgraphs are shown in Table 8.1.

8.2.2 Encoding

Based on the constraints proposed in Sect. 8.1.3, the method in reference [3] was used for preliminary encoding. For the graph G shown in Fig. 5.4, 129 DNA sequences representing colors were encoded.

At the same time, according to the method of constructing probes in Sect. 8.1.4, and the number of probes that need to be constructed for the probe graph corresponding to each subgraph is 185, denoted as $\overline{x_i x_{i+1}}$, where $x_i \in C_i$, $x_{i+1} \in C_{i+1}$, and $x_i \neq x_{i+1}$.

8.2.3 Construction of Initial Solution Space

Here, taking the subgraph G_1 as an example to introduce the construction method of the initial solution space and the detection results. First, the 5′ end of the 33 DNA sequences representing the vertex color is phosphorylated, and then annealed with the corresponding 46 probes. The reaction conditions are: 94 °C, 5 min, 50 °C, 10 min. The reaction product is ligated with T4 DNA ligase and left overnight at 16 °C. Finally, the primer pair $< r_1, \overline{b_{16}} >$ is used for amplification, and a DNA sequence set of 320 bp size is obtained, which is $S(G_1)$ (Fig. 8.5a). The product is recovered and dissolved in 50 μl of sterile water, and its concentration is measured to be 400 ng/μl. The construction methods of $S(G_2)$, $S(G_3)$, and $S(G_4)$ are the same, and the experimental results are shown in Fig. 8.5b–d.

Afterwards, $< r_1, \overline{x_i} >$ is used, where $x = \{r, b, y\}$, $i = 2, 3, \cdots, 16$, as the primer for the PCR reaction to judge the completeness of the initial solution space. The PCR reaction system and conditions are the same as above. The detection results are shown in Fig. 8.5e–f. The experimental results show that the vertex position in the DNA sequence is consistent with the vertex order of the corresponding subgraph, and each vertex's possible coloring scheme exists in the initial solution space.

8.2.4 Subgraph Non-solution Deletion

(1) Paths or Circles Formed by Forward Edges

The deletion of non-solution can be performed in parallel according to the multi-edge constraint, greatly reducing the number of PCR operations. The experimental results are given using the path $P = 16 - 20 - 22 - 26 - 30 - 31$ in subgraph G_2 as an example.

First, using $< b_{16}, \overline{r_{20}} >$, $< b_{16}, \overline{y_{20}} >$, $< r_{20}, \overline{y_{22}} >$, $< y_{20}, \overline{r_{22}} >$, $< r_{22}, \overline{y_{26}} >$, $< y_{22}, \overline{r_{26}} >$, $< r_{26}, \overline{y_{30}} >$, $< r_{26}, \overline{b_{30}} >$, $< y_{26}, \overline{b_{30}} >$, $< y_{30}, \overline{r_{31}} >$, and $< b_{30}, \overline{r_{31}} >$ as primer pairs, the DNA in the initial solution space

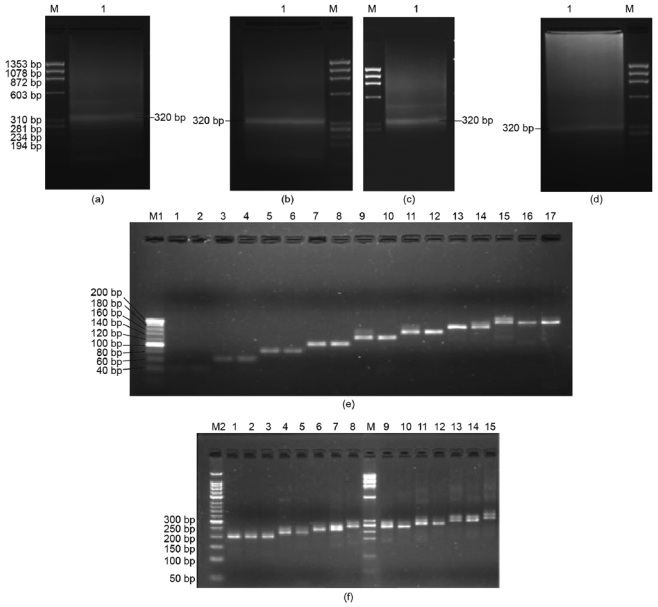


Fig. 8.5 Construction and detection of the initial solution space, where M is the DNA marker ϕ X174-Hae III. The lanes 1 in (**a**)–(**d**) are the initial solution spaces of each constructed subgraph, where (**a**)–(**d**) correspond to $S(G_1)$, $S(G_2)$, $S(G_3)$, and $S(G_4)$, respectively, each DNA sequence is 320 bp in size. (**e**) and (**f**) are the detection results of the solution library [1]

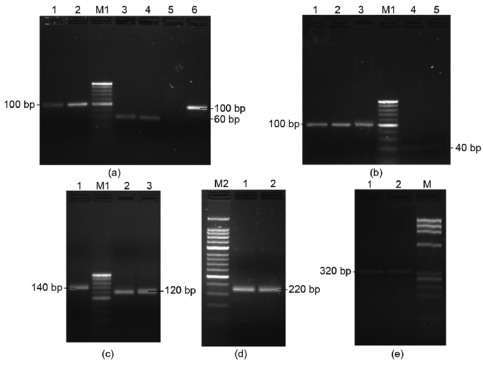


Fig. 8.6 Experimental results of non-solution deletion for the circle $C = 16 - 17 - \cdots - 31 - 16$ [1]

is used as a template to decompose the full-length 320 bp DNA sequence into several fragments of different sizes: 100 bp (16–20), 60 bp (20–22), 100 bp (22–26), 100 bp (26–30) and 40 bp (30–31) (see Fig. 8.6a–b). It can be seen that there is no band in the lane 5 of Fig. 8.6a which is according to the primer pair $< r_{22}, \overline{y_{26}} >$. That means there is no DNA sequence containing both r_{22} and y_{26} when constructing the initial solution space. This further proves through experiments that some non-solutions are deleted when constructing the initial solution space.

Next, the second PCR operation for this circle is performed. Using $< b_{16}, \overline{y_{22}} >$ as the primer, the mixture of PCR products in lanes 1 and 3 of Fig. 8.6a is used as the template, and a 140 bp electrophoresis band is obtained; using $< r_{26}, \overline{r_{31}} >$ as the primer, the mixture of PCR reaction products in lanes 1 and 4 of Fig. 8.6b is used as the template, and a 120 bp electrophoresis band is obtained; using $< r_{26}, \overline{r_{31}} >$ as the primer, the mixture of PCR reaction products in lanes 2 and 5 of Fig. 8.6b is used as the template, and a 120 bp electrophoresis band is obtained. The electrophoresis results are shown in Fig. 8.6c.

Again, the third PCR operation for this circle is performed. Using $< b_{16}, \overline{r_{26}} >$ as the primer, the mixture of PCR products in lane 6 of Fig. 8.6a and lane 1 of Fig. 8.6c is used as the template, and a 220 bp electrophoresis band is obtained. The electrophoresis results are shown in Fig. 8.6d.

Finally, the fourth PCR operation for this circle is performed. Using $< b_{16}, \overline{r_{31}} >$ as the primer, the mixture of PCR products in lane 1 of Fig. 8.6d and lane 2 of Fig. 8.6c is used as the template; the mixture of PCR products in lane 1 of Fig. 8.6d and lane 3 of Fig. 8.6c is used as the template, and two DNA strands with 320 bp are obtained, respectively. The electrophoresis results are shown in Fig. 8.6e.

After the above reactions are completed, two sets of solutions are generated: $b_{16}r_{20}y_{22}r_{26}y_{30}r_{31}$ and $b_{16}r_{20}y_{22}r_{26}b_{30}r_{31}$. These two sets of solutions will serve as new templates for the next non-solution deletion process.

(2) Single Edges with Undetermined Coloring of Associated Vertices

The experimental results are given using the edge $e = \{4, 8\}$ in subgraph G_1 as an example.

Firstly, using the product of the previous PCR as a template, the first PCR was performed with $< r_1, \overline{r_4} >$, $< r_4, \overline{y_8} >$, $< y_8, \overline{b_{16}} >$, $< r_1, \overline{y_4} >$, $< y_4, \overline{r_8} >$, $< r_8, \overline{b_{16}} >$ as primer pairs, the results are shown in Fig. 8.7a. Among them, when $< y_4, \overline{r_8} >$ is used as the primer, no PCR product is generated.

Secondly, using $< r_1, \overline{y_8} >$ as the primer, and the mixture of PCR products from lanes 1 and 3 in Fig. 8.7a as the template, the second PCR operation was performed. A DNA sequences with 160 bp were obtained (shown in Fig. 8.7b).

Finally, using $< r_1, \overline{b_{16}} >$ as the primer, and the mixture of PCR products from lane 5 in Fig. 8.7a and lane 1 in Fig. 8.7b as the template for the PCR reaction, DNA strands with 320 bp were obtained (see Fig. 8.7c). At this point, the three PCR operations for $e = \{4, 8\}$ are completed and the coloring schemes representing vertices 4 and 8 in the DNA sequence have been determined to be r_4, y_8.

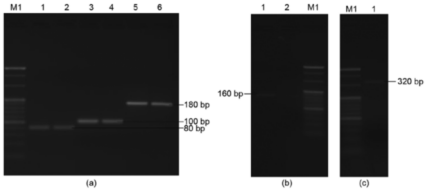


Fig. 8.7 Experimental results of deleting non-solutions for edge $e = \{4, 8\}$ [1]

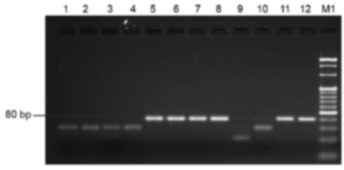


Fig. 8.8 Experimental results of deleting non-solutions for edge $e = \{34, 39\}$ [1]

(3) Edges with Vertices Whose Coloring Has Been Determined
The experimental results are given using the edge $e = \{39, 42\}$ of subgraph G_3 as an example, as shown in Fig. 8.8. When all PCR reactions for deleting circle $C_1 = 31 - 35 - 39 - 43 - 31$ in subgraph G_3 are completed, the coloring of vertex 39 is determined. After the first PCR reaction of circle $C_2 = 34 - 38 - 42 - 46 - 34$, the coloring of vertex 42 is determined. At this time, while deleting non-solutions of circle C_2, the exclusion method can be used to delete non-solutions for $e = \{39, 42\}$. Using the primer pair $< y_{39}, \overline{y_{42}} >$, and the 6 DNA sequences produced after the above reaction as templates, PCR reaction is performed. In Fig. 8.8, lanes 5, 6, 7, 8, 11, 12 show an 80 bp electrophoresis band, indicating that these corresponding DNA sequences represent the situation where vertices 39 and 42 are colored the same, and need to be discarded. The templates that did not form a length of 80 bp represent the situation where vertices 39 and 42 are colored normally, and will be retained for subsequent non-solution deletion operations.

8.2.5 Subgraph Merging and Non-solution Deletion

After completing the parallel operations for each subgraph, the solutions of each subgraph need to be merged and further non-solutions deleted. The mixtures of DNA sequences of length 320 bp representing solutions that satisfy normal 3-coloring of subgraphs G_1 and G_2 after the above reactions are mixed pairwise as templates. Using $< r_1, \overline{r_{31}} >$ as primers, PCR amplification is performed to obtain DNA stands of length 620 bp, which are all possible 3-coloring solutions of subgraph $G[V_1 \cup V_2]$ derived from graph G , that is, the initial solution space of subgraph $G[V_1 \cup V_2]$. Then, the non-solutions in this initial solution space are completely deleted using the method of deleting non-solutions of the above subgraphs. The results obtained are denoted as X_i, $i = 1, 2, 3, 4$, see Fig. 8.9a, b.

$X_1 = r_1 b_2 y_3 r_4 y_5 b_6 r_7 y_8 r_9 b_{10} y_{11} r_{12} b_{13} y_{14} r_{15} b_{16} r_{17} y_{18} b_{19} r_{20} b_{21} y_{22} r_{23} b_{24} y_{25} r_{26} y_{27} b_{28} r_{29} y_{30} r_{31}$;
$X_2 = r_1 b_2 y_3 r_4 y_5 b_6 r_7 y_8 r_9 b_{10} y_{11} r_{12} b_{13} y_{14} r_{15} b_{16} r_{17} b_{18} y_{19} r_{20} b_{21} y_{22} r_{23} b_{24} y_{25} r_{26} y_{27} b_{28} r_{29} y_{30} r_{31}$;
$X_3 = r_1 b_2 y_3 r_4 y_5 b_6 r_7 y_8 b_9 y_{10} r_{11} y_{12} r_{13} b_{14} y_{15} b_{16} r_{17} y_{18} b_{19} r_{20} b_{21} y_{22} r_{23} b_{24} y_{25} r_{26} y_{27} b_{28} r_{29} y_{30} r_{31}$;

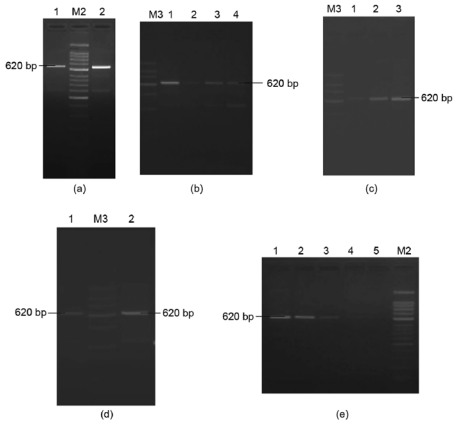


Fig. 8.9 Solution of subplots $G[V_1 \cup V_2]$ and $G[V_3 \cup V_4]$. (**a**) The PCR products of lanes 1, 2 are X_1 and X_2; (**b**) the PCR products of lanes 1, 2 are all X_3, and the PCR products of lanes 3, 4 are all X_4. Their primer pairs are all $\langle r_1, \overline{r_{31}} \rangle$. (**c**) The products of lanes 1, 2 are all Y_1; and the products of lane 3 are all Y_2; (**d**) the product of lane 1 is Y_3; and lane 2 the product of lane 1 is Y_4; in (**e**) the product of lanes 1, 2 is Y_5; the product of lanes 3, 4 is Y_6. Their primer pairs are all $\langle r_{31}, \overline{y_{61}} \rangle$

$X_4 = r_1 b_2 y_3 r_4 y_5 b_6 r_7 y_8 b_9 y_{10} r_{11} y_{12} r_{13} b_{14} y_{15} b_{16} r_{17} b_{18} y_{19} r_{20} b_{21} y_{22} r_{23} b_{24} y_{25} r_{26} y_{27} b_{28} r_{29} y_{30} r_{31}$.

At the same time, the same method is used to process subgraphs G_3 and G_4, and the results obtained are denoted as Y_i, $i = 1, 2, 3, 4, 5, 6$, see Fig. 8.9c, d and e.

$Y_1 = r_{31} b_{32} y_{33} r_{34} b_{35} y_{36} r_{37} b_{38} y_{39} r_{40} y_{41} r_{42} b_{43} y_{44} r_{45} b_{46} y_{47} b_{48} r_{49} y_{50} b_{51} r_{52} y_{53} b_{54} r_{55} y_{56} r_{57} b_{58} y_{59} r_{60} y_{61}$;

$Y_2 = r_{31} b_{32} y_{33} r_{34} b_{35} y_{36} r_{37} b_{38} y_{39} r_{40} y_{41} r_{42} b_{43} y_{44} r_{45} b_{46} r_{47} b_{48} r_{49} y_{50} b_{51} r_{52} y_{53} b_{54} r_{55} y_{56} r_{57} b_{58} y_{59} b_{60} y_{61}$;

$Y_3 = r_{31} b_{32} y_{33} r_{34} b_{35} y_{36} r_{37} b_{38} y_{39} r_{40} y_{41} r_{42} b_{43} y_{44} r_{45} b_{46} r_{47} b_{48} r_{49} y_{50} b_{51} r_{52} y_{53} b_{54} r_{55} y_{56} r_{57} b_{58} y_{59} b_{60} y_{61}$;

$Y_4 = r_{31} b_{32} y_{33} r_{34} b_{35} y_{36} r_{37} b_{38} y_{39} b_{40} y_{41} r_{42} b_{43} y_{44} r_{45} b_{46} y_{47} b_{48} r_{49} y_{50} b_{51} r_{52} y_{53} b_{54} r_{55} y_{56} r_{57} b_{58} y_{59} r_{60} y_{61}$;

$Y_5 = r_{31} b_{32} y_{33} r_{34} b_{35} y_{36} r_{37} b_{38} y_{39} b_{40} y_{41} r_{42} b_{43} y_{44} r_{45} b_{46} r_{47} b_{48} r_{49} y_{50} b_{51} r_{52} y_{53} b_{54} r_{55} y_{56} r_{57} b_{58} y_{59} b_{60} y_{61}$;

$Y_6 = r_{31} b_{32} y_{33} r_{34} b_{35} y_{36} r_{37} b_{38} y_{39} b_{40} y_{41} r_{42} b_{43} y_{44} r_{45} b_{46} r_{47} b_{48} r_{49} y_{50} b_{51} r_{52} y_{53} b_{54} r_{55} y_{56} r_{57} b_{58} y_{59} b_{60} y_{61}$.

Mix the DNA sequence with a full length of 620 bp as a template, use $< r_1, \overline{y_{61}} >$ as the primer pair, perform PCR amplification, and obtain a DNA sequence set with a length of 1220 bp, representing the feasible solution library of graph G. Use the same method to delete non-solutions, and the final DNA sequence represents the solution that satisfies the normal 3-coloring of graph G. There are 8 DNA sequences, each representing 8 solutions (see Fig. 8.10). After sequencing,

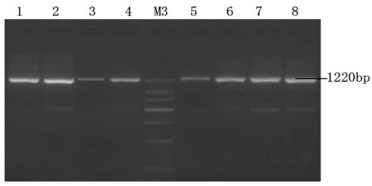


Fig. 8.10 The solution of the graph G, M3 is the DNA marker of 150 bp ladder. Lanes 1–8 correspond to solutions $Z_1 - Z_8$ [1]

the solution that satisfies the normal 3-coloring of graph G is obtained, denoted as $Z_i, i = 1, 2, \cdots, 8$.

$Z_1 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}y_{18}b_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}$
$y_{27}b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}r_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}$
$r_{52}y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}r_{60}y_{61}$;

$Z_2 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}y_{18}b_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}r_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}b_{60}y_{61}$;

$Z_3 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}y_{18}b_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}b_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}r_{60}y_{61}$;

$Z_4 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}y_{18}b_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}b_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}b_{60}y_{61}$;

$Z_5 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}b_{18}y_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}r_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}r_{60}y_{61}$;

$Z_6 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}b_{18}y_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}r_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}b_{60}y_{61}$;

$Z_7 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}b_{18}y_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}b_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}r_{60}y_{61}$;

$Z_8 = r_1b_2y_3r_4y_5b_6r_7y_8r_9b_{10}y_{11}r_{12}b_{13}y_{14}r_{15}b_{16}r_{17}b_{18}y_{19}r_{20}b_{21}y_{22}r_{23}\ b_{24}y_{25}r_{26}y_{27}$
$b_{28}r_{29}y_{30}r_{31}b_{32}y_{33}r_{34}b_{35}y_{36}r_{37}b_{38}y_{39}b_{40}y_{41}r_{42}b_{43}y_{44}r_{45}\ b_{46}y_{47}b_{48}r_{49}y_{50}b_{51}r_{52}$
$y_{53}b_{54}r_{55}y_{56}r_{57}b_{58}y_{59}b_{60}$.

8.3 Complexity Analysis

This section conducts a theoretical analysis of the model from the perspectives of reducing the complexity of the initial solution space and enhancing the parallelism of PCR operation methods. To facilitate comprehension, the calculation formula for the DNA strands is introduced first. This formula can be employed to compute the number of paths between any two vertices of a given graph, thus enabling path counting.

Lemma 8.1 *Assume G is a n subgraph, and $V(G) = \{v_1, v_2, \cdots, v_n\}$. A is the adjacency matrix of G. Then the number of paths from vertex v_i to v_j with a length of l is the value at the (i, j) position in the matrix A^l . Here A^l is the matrix obtained after l times A multiplication [4].*

8.3.1 Analysis of Reducing the Complexity of the Initial Solution Space

In order to reduce the initial solution space of the DNA computing model for solving the graph vertex coloring problem, two methods are used in the model, which are the subgraph partitioning method and the subgraph vertex color set determination method given in Sects. 8.1.1 and 8.1.2. The introduction of the probe graph includes all the information of these two methods. Using the probe graph, a calculation formula for the number of different DNA sequences in the initial solution space in the model can be further given. This formula can be given by the following theorem:

Theorem 8.2 *Assume G is a n order graph, G_1 is a subgraph of graph G. If $V(G_1) = \{v_1, v_2, \cdots, v_t\}$, and vertices v_1 and v_t are its two bridge vertices. Let $B(G)$ represent the probe graph of G_1. Then the number of different DNA stands in the initial solution space of G_1 is the number of paths from vertex v_1 to vertex v_t with a length of $t-1$ in graph $B(G)$, denoted as $N_p(v_1, v_t)$, given by the following formula:*

$$N_p(v_1, vt) = A^{t-1}(B(G_1))(1, t-1) \tag{8.4}$$

Proof From the definition of the probe graph $B(G)$, it could be known that the vertex x_i in $B(G_1)$ represents the color that the vertex v_i in the graph can be colored, and the edge $\{x_i, z_{i+1}\}$ of $B(G_1)$ represents the probe formed between x_i and z_{i+1}, that is, it indicates that in the initial solution space, when the vertex v_i is colored as x , vertex v_{i+1} can be colored z , $x, z \in \{r, y, b\}$. Therefore, any path from vertex r_1 to vertex b_t in $B(G_1)$ represents a coloring of subgraph G_1, and all paths from vertex r_1 to vertex b_t represent all possible solutions in the initial solution space.

Further examining the structure of $B(G_1)$, the graph can be divided into t groups, where the subscript i represents the i group. From the definition of $B(G_1)$, it could be know that the i group is only connected to the $i+1$ group, and there must be a connection. Therefore, in the probe graph $B(G_1)$, the length of each path from vertex r_1 to vertex b_t can only be $t-1$. Therefore, by Lemma 8.1, this theorem is proven. □

Corollary 8.1 *The number of different DNA stands in the initial solution space of subgraphs G_1, G_2, G_3 and G_4 shown in Fig. 8.2 are: 89, 81, 412 and 151 respectively; the proportions of non-solutions deleted from the enumeration-type initial solution space are: 99.9998% , 99.9998% , 99.999% and 99.9997% .*

Proof From Theorem 8.2, it could be known that the number of DNA stands in the initial solution space of graph G_1 is the number of paths of length 15 between vertices r_1 and b_{16} in the probe graph B_1 (see Fig. 8.4) is $N_{33}(r_1, b_{16})$.

In order to calculate $N_{33}(r_1, b_{16})$, first use Lemma 8.1 to give the adjacency matrix $A(B_1)$ of graph B_1.

$$A(B_1) = \begin{bmatrix}
0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&1&0&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&1&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&1&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&1&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&1&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&1&1&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&0&1&1
\end{bmatrix}$$

The correspondence between the labels of rows and columns in the adjacency matrix and the vertices in the probe graph B_1 is as follows:

$$\begin{pmatrix}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17\\
r_1 & y_2 & b_2 & r_3 & y_3 & r_4 & y_4 & y_5 & b_5 & y_6 & b_6 & r_7 & y_7 & r_8 & y_8 & r_9 & y_9\\
18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 & 33 & \\
b_9 & r_{10} & y_{10} & b_{10} & r_{11} & y_{11} & r_{12} & y_{12} & r_{13} & y_{13} & b_{13} & y_{14} & b_{14} & r_{15} & y_{15} & b_{16} &
\end{pmatrix}$$

It is easy to calculate that $A(B_1)^{15}(1, 33) = 89$. This shows that for the 3-coloring of subgraph G_1, the number of different DNA stands in the initial solution

space representing all possible 3-colorings is 89. Compared with the DNA stands $3^{16} = 43046721$ required by enumeration: $\frac{89}{43046721} \approx 0.000002 = 0.0002\%$. That is to say, when constructing the initial solution space of subgraph G_1 , 99.9998% of non-solutions were deleted. Similarly, for subgraph G_2, $A(B_2)^{15}(1, 32) = 81$; for subgraph G_3, $A(B_3)^{15}(1, 35) = 412$; for subgraph G_4, $A(B_4)^{15}(1, 32) = 151$. This shows that in the construction of $S(G_2)$, $S(G_3)$ and $S(G_2)$, 99.9998% , 99.999% and 99.9997% of non-solutions were deleted respectively. □

8.3.2 *Enhancing Parallelism with PCR Technology*

The following provides a detailed analysis of the mechanism by which the use of parallel PCR technology can greatly reduce the number of biological operations. First, a general method and steps for deleting non-solutions using parallel PCR are given.

Let $P = v_{i0} \cdots v_{im}$ is a directed path in the subgraph G_1 with a vertex count of $m + 1$ and an edge count of m. Because the entire subgraph must be considered during parallel PCR, two bridge vertices v_1 and v_t must be considered during the PCR operation. Therefore, there are the following three cases:

Case 1 The two endpoints of the path are not bridge vertices of the subgraph, that is, $v_{i0} \neq v_1$ and $v_{im} \neq v_t$; for this case, these two bridge vertices v_1 and v_t need to be added, as shown in Fig. 8.11a.

Case 2 Exactly one of the two endpoints of the path is a bridge point of the subgraph, that is, either $v_{i0} = v_1$ and $v_{im} \neq v_t$ or $v_{i0} \neq v_1$ and $v_{im} = v_t$; for this case, one bridge point v_t or v_1 needs to be added, as shown in Fig. 8.11b.

Case 3 Both endpoints of the path are bridge vertices of the subgraph, that is, either $v_{i0} = v_1$ and $v_{im} = v_t$; for this case, only the path P is considered, as shown in Fig. 8.11c.

In fact, the above cases 2 and 3 can be considered as special cases of case 1. That is, for cases 2 and 3, delete the edge adjacent to the bridge point: from the path P in case 2, delete an edge v_{i0}, v_{i1} or $v_{i(m-1)}$, v_{im} to get a path similar to case 1 P' ; from the path P in case 3, delete two edges v_{i0}, v_{i1} and $v_{i(m-1)}$, v_{im} to get a path

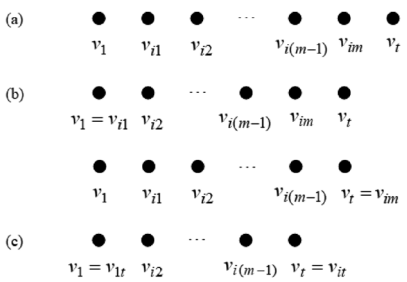


Fig. 8.11 Three cases of directed paths and bridge vertices [1]

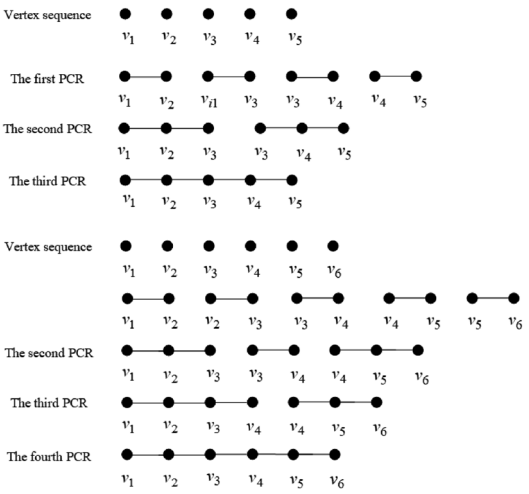


Fig. 8.12 Schematic diagram of the number of parallel PCR operations [1]

similar to case 1 P'' . Therefore, only the path in case 1 is considered to introduce the parallel PCR method.

According to the method in Fig. 8.11 to determine the number and order of vertices in the directed path $P = v_{i0}v_{i2}\cdots v_{im}$, it is known that the number of PCR amplifications is related to the number of vertices in the path P. The first PCR operation is mainly to obtain the correct coloring scheme, and the primer pairs required are: $< v_1, v_{i0} >$, $< v_{i0}, v_{i1} >$, $\cdots$, $< v_{i(m-1)}, v_{im} >$, $< v_{im}, v_t >$; the second PCR is to connect the two adjacent fragments after the first PCR reaction; the third is to connect the two adjacent fragments after the second PCR reaction, and so on. Here, the cases with 5 vertices (i.e., two-edge paths) and 6 vertices (i.e., three-edge paths) are explained (Fig. 8.12), where the connecting edges represent the parallel PCR amplification operations performed between two or more vertices.

Theorem 8.3 *Let P be a directed path in the subgraph G_1 with an edge count of x. When $2^{l-1} < x \leq 2^l$, define $\langle x \rangle = 2^l$. The number of parallel PCR operations, denoted as $PCR(x)$, should be*

$$PCR(x) = 1 + log_2\langle x+2 \rangle, \quad x \geq 1 \tag{8.5}$$

Proof For case 1, the two endpoints of the path are not bridge vertices of the subgraph, that is, $v_{i0} \neq v_1$ and $v_{im} \neq v_t$; if the path P contains x edges, then the number of vertices in the path is $x + 1$, since this path does not contain bridge vertices v_1, v_t, according to the parallel PCR method, two more bridge vertices need to be added, making the number of vertices $x+3$ (see Fig. 8.11a). Suppose $x+2 = 2^l$, the number of vertex pairs in the first PCR operation is the number of edges after adding the bridge vertices v_1 and v_t, which is $(x + 3) - 1 = x + 2 = 2^l$. According to the definition of parallel PCR, the number of vertex pairs for the second operation is $(x + 2)/2$, 2^{l-1} , the number of pairs for the third PCR operation is 2^{l-2}, $\cdots$,

the number of pairs for the l PCR operation is $2^{l-(l-1)} = 2$, the $l + 1$ operation can complete all PCR operations, that is, it is inferred: when $x + 2 = 2^l$, the number of PCR operations $l + 1$. So $PCR(x) = l + 1 = 1 + \log_2 2^l = 1 + \log_2\langle 2^l\rangle = 1+\log_2\langle x+2\rangle$. This proves that when $x+2 = 2^l$, the conclusion holds. Similarly, it can be proved that when $x+2 = 2^{l+1}$, the conclusion holds, that is, $PCR(x) = l+2$.

Next, let's prove the case of $x + 2 = 2^l + 1$.

Similar to the above proof, the number of vertices of the path P is $x + 1$, plus two bridge vertices, making the number of vertices $x + 3$, when $x + 2 = 2^l + 1$, that is, $x + 3 = 2^l + 2$ vertices, therefore, the number of pairs for the first PCR operation is $(x + 3) - 1 = x + 2 = 2^l + 1$; the number of pairs for the second operation is $2^{l-1} + 1$, the number of pairs for the third PCR operation is $2^{l-2} + 1$, $\cdots$, the number of pairs for the l PCR operation is $2^{l-(l-1)} + 1 = 3$, the number of operations for the $l + 1$ operation is 2, the $l + 2$ can complete all PCR operations. On the other hand, substituting $x + 2 = 2^l + 1$, we have: $PCR(x) = l + 2 = 1 + (l+) = 1 + \log_2 2^{l+1} = 1 + \log_2\langle 2^l + 1\rangle = 1 + \log_2\langle x + 2\rangle$ This proves that when $x + 2 = 2^l + 1$, the conclusion holds.

Since for any $2 \le y < 2^l$, $x + 2 = 2^l + y$, the number of PCR operations corresponding to it is not less than $x + 2 = 2^l + 1$ corresponding to the PCR operation, but not more than $x + 2 = 2^{l+1}$ corresponding to the PCR operation, and both of these PCR operation times are $l + 2$, this proves that for any $2 \le y < 2^l$, $x + 2 = 2^l + y$, the conclusion holds.

As mentioned above, the second and third situations in Fig. 8.12 can eventually be converted into the first situation, so this theorem is proved. □

Theorem 8.3 fully characterizes the relationship between the length of the forward path x and the number of PCR operations during parallel PCR operations. Obviously, parallel PCR operations can greatly reduce the number of operations, greatly improving the running speed of the DNA computer. Table 8.2 and the curve in Fig. 8.13 shows the advantages of parallel PCR. In Table 8.2, x represents the number of edges, $3x$ and $PCR(x)$ respectively represent the number of PCR operations corresponding to the edge deletion and parallel PCR methods.

Table 8.2 Comparison of the number of PCR operations between edge deletion and parallel PCR methods

x	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$PCR(x)$	3	3	4	4	4	4	5	5	5	5	5	5	5	5	6
$3x$	3	6	9	12	15	18	21	24	27	30	33	36	39	42	45
x	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
$PCR(x)$	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
$3x$	48	51	54	57	60	63	66	69	72	75	78	81	84	87	90

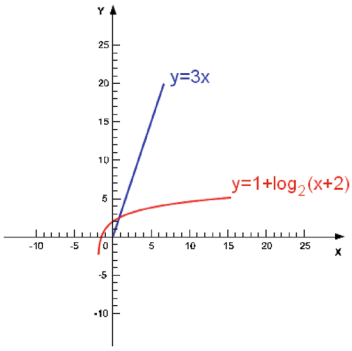


Fig. 8.13 Curve of the number of PCR operations with edge-by-edge deletion and parallel PCR methods, where x represents the number of edges in the forward path P, y represents the number of PCR operations [1]

Corollary 8.2 *Assume P is a forward path in the subgraph G_1 with x edges, then the following conclusion can be drawn:*

$$\lim_{x\to\infty} \frac{log_2\langle x+2\rangle + 1}{3x} = 0 \tag{8.6}$$

Proof Let $x + 2 = 2^l + y$, $1 \le y < 2^l$. Then $\langle x + 2\rangle = 2^{l+1} = 2(2^l + y - y) < 2(2^l + y)$; also because $2^l > y$, $2^l + 2^l > y + 2^l$, that is, $2^{l+1} > y + 2^l$, thus we have: $y + 2^l < 2^{l+1} < 2(2^l + y)$, this is

$$x + 2 < \langle x + 2\rangle < 2(x + 2)$$

thus we have

$$log_2(x + 2) < log_2\langle x + 2\rangle < log_2 2(x + 2)$$

Therefore,

$$\frac{1 + log_2(x + 2)}{3x} < \frac{1 + log_2\langle x + 2\rangle}{3x} < \frac{1 + log_2 2(x + 2)}{3x},$$

By L'Hopital's rule, we have

$$\lim_{x\to\infty} \frac{log_2(x + 2) + 1}{3x} = \lim_{x\to\infty} \frac{\log_2 2(x + 2) + 1}{3x} = 0$$

Thus, this corollary is proven. □

Corollary 8.2 indicates that as the number of edges in the forward path P in the subgraph increases, the advantage of the parallel PCR method becomes greater, that is, the speed of DNA computation becomes faster. Since each PCR takes a long time, about half an hour, this further highlights the advantage of the parallel PCR method.

The article discusses in detail the DNA computing model for parallel graph vertex coloring. This model uses the decomposition and merging of graphs and the use of parallel PCR technology, not only greatly reducing the complexity of the initial solution space constructed, but also greatly reducing the complexity of biological computation. The parallel DNA computing model is designed around how to overcome the problem of exponential explosion of the solution space and improve the scale of computation. It can not only eliminate 99% of the non-feasible solutions when constructing the initial solution space, but also use DNA self-assembly and parallel PCR methods to obtain solutions through identification, splicing, and assembly techniques. This model can be used to solve the graph coloring problem with 61 vertices. This example is also the largest scale problem solved by DNA computation to date. However, as the scale of the problem continues to increase, the number of basic DNA sequences required will also increase, which involves the encoding problem mentioned in Chap. 5.

References

1. Xu, J., Qiang, X., Zhang, K. et al.: A DNA computing model for the graph vertex coloring problem based on a probe graph. Engineering **4**: 61–77 (2018).
2. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(5187): 1021–1024 (1994).
3. Zhang, K., Pan, L., Xu, J.: A global heuristically search algorithm for DNA encoding. Progress in Natural Science **17**(6):745–9 (2017).
4. Biggs, N.: Algebraic graph theory. Cambridge University Press, Cambridge (1993).

Chapter 9
Probe Machine

This chapter introduces an underlying fully parallel computing model—the probe machine [1]. Unlike traditional Turing machines, the probe machine overcomes the limitations of sequential computation by enabling any two data to process information simultaneously without relying on linear adjacency. This fully parallel computing mechanism makes the probe machine highly efficient in solving complex problems, typically requiring only one or a few probe operations to obtain all solutions. The design of the probe machine emphasizes efficient parallelism, and its computational power significantly surpasses that of traditional models, particularly in solving large-scale problems. This chapter will introduce the probe machine from six perspectives: the background of its creation, its fundamental principles, typical applications, biological computation implementation, the relationship between the probe machine and biological neural networks, and an analysis of the machine's functions. The goal is to highlight its scientific value and potential applications.[1]

9.1 Background of the Probe Machine

Computing tools are indispensable instruments in human society and civilization, continuously evolving alongside human development progress. Human civilization has progressed through five key stages: the Stone Age, the Iron Age, the Steam Age, the Electric Age, and the Information Age. During each stage, computing tools have evolved significantly—from simple to complex, from low-level to high-level. Beginning with rudimentary methods such as knotting records, counting rods, and abacuses, and progressing through slide rules, mechanical computers, and ultimately

[1] The content presented in this chapter is largely based on the author's previously published work [1], with permission from IEEE Transactions on Neural Networks and Learning Systems, Jin Xu, "Probe Machine", 2016; all rights reserved.

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_9

to today's electronic computers, these tools have played crucial roles in advancing various historical epochs.

Conceptually, a computer is a machine made from materials that can implement a computing model. For example, the computing model of today's electronic computer is the Turing machine [2], and the material that implements this model is electronic components [3]. The Turing machine is a mathematical model established by Turing to solve the "decidability of the solvability of Diophantine equations". This problem is the 10th of the 23 problems proposed by Hilbert at the International Congress of Mathematicians held in Paris, France in August 1900. The Turing machine consists of a finite controller, an infinitely extendable tape, and a read-write head that can move left and right on the tape.

In 1945, von Neumann established the electronic computer system architecture with the Turing machine as the computing model, using diodes, triodes, etc. as the main materials to implement this model. In February 1946, the first general-purpose electronic digital computer ENIAC based on this system architecture was successfully developed. To this day, the development of electronic computers has mainly gone through four stages: electronic tube digital computers, transistor digital computers, integrated circuit digital computers, and large-scale integrated circuit computers.

Research has found that computing devices based on Turing machines (such as electronic computers) astonishingly follow Moore's Law [4] in their development process. Therefore, scientists have explored new computing models for half a century to develop more powerful computers. In 2011, on the 100th anniversary of Turing's birth, a global call was made for new computing models that surpass the Turing machine. The motivations mainly include two aspects: First, the manufacturing technology of electronic computers is about to reach its limit, as the famous theoretical physicist Kaku predicted in 2012 that the manufacturing technology of semiconductors will reach its limit within ten years; Second, due to the Turing machine model, electronic computers have always been unable to handle large-scale **NP**-complete problems.

Various approaches such as bionic computing (including neural networks, evolutionary computing, and particle swarm optimization), optical computing, quantum computing, and biological computing have been proposed in exploring new computational models. Both the bionic and optical computing models are comparable to the Turing machine's computational power. This is because bionic computing is carried out using electronic computers. Although optical computing relies on optical devices for implementation, its computational model is still fundamentally based on the Turing machine [5–7].

The best effect of quantum computing in dealing with **NP**-complete problems is: if the complexity of a certain algorithm under the Turing machine is n, then under the quantum computing model, its complexity can be reduced to $\sqrt{n}$ [8–10]. Therefore, the quantum computing model has not actually surpassed the Turing machine model. In contrast, in the research of biological computing, people seem to have seen the dawn of surpassing the Turing machine. As a nanoscale DNA molecule for data, the huge parallelism during specific hybridization makes it possible for DNA

computing to quickly solve certain scale **NP**-complete problems. Early models of DNA computing include the computing models proposed by Adleman and others [11], the paste model [12], the self-assembly model [13–15], the non-enumerative DNA computing model [16], and parallel DNA computing model [17]. These models use nucleic acid molecules' characteristics to increase computing speed greatly, but there are still many non-solutions in the computing process. In recent years, many computing models/architectures based on neural networks have been proposed to solve complex problems. For example, Hu and others [18] proposed a hardware-based training scheme to alleviate (or even eliminate) most of the noise. Duan and others [19] introduced a honeycomb nonlinear/neural network based on memristors. Gong and others [20] proposed a multi-objective sparse feature learning model for deep neural networks. Xia and others [21] proposed a dual-projection neural network to solve a class of constrained quadratic optimization problems.

It has been proven that the above computing models are equivalent to the Turing machine in terms of computability, but different models will lead to different computational effectiveness.

9.2 Principle of Probe Machine

Before introducing the mathematical model of the probe machine, let's first analyze the mechanism of the Turing machine.

9.2.1 Analysis of Turing Machine Mechanism

Consider the following two challenging questions: (1) Why do computers based on Turing machines always generate many non-solutions in solving **NP**-complete problems? (2) Why are all existing computing models equivalent to the Turing machine? We answer these two questions by characterizing the mechanism of the probe machine from two aspects.

9.2.1.1 Linear Data Placement Mode

Existing computing devices rely on traditional data storage methods, known as the data placement mode, during their computation process. Whether it is today's electronic computers or future generations of computing devices, the data placement modes employed can generally be classified into three types: one-dimensional, two-dimensional, and three-dimensional. For example, the data in electronic computers is stored in a linear fashion, making its data placement mode one-dimensional. In contrast, the data placement mode of the probe machine, which will be introduced later, is three-dimensional.

Throughout the development of human civilization over thousands of years, whether in writing or recording numbers, data has typically been recorded sequentially—either from left to right or from top to bottom. This means that data units have been placed or stored linearly. Influenced by this "mode of thinking," all computing tools created by humans, from the earliest methods of knotting records and stone carvings to the abacus and modern Turing machine-based electronic computers, have followed this linear approach for storing and processing data. We refer to this sequential data arrangement as the linear placement mode. Under this mode, only adjacent data units can interact with each other, which significantly constrains the potential of the data and limits the computational power of these tools. All of the computing models mentioned earlier are subject to this limitation. For instance, when a computer program attempts to solve an **NP**-complete problem, the initial solution space often contains many suboptimal solutions, making it akin to searching for a needle in a haystack. The linear placement mode is the underlying cause of this large pool of non-optimal solutions, which hinders the search for the true solution.

9.2.1.2 Serial Data Processing Mode

A Turing machine-based computing tool processes only two adjacent data units during each computation in the linear data placement mode. We refer to this mode of data processing as serial. The serial data processing mode contributes to the presence of many non-optimal solutions in the initial solution space when solving problems.

Since bionic computing, quantum computing, and DNA computing models all employ linear data placement and serial data processing modes, similar to Turing machines, these computing models are essentially equivalent to Turing machines.

The previous discussion highlights that the two main factors limiting the computational power of the Turing machine are the linear data placement and serial data processing modes. Therefore, these two constraints must be overcome to achieve a computational model with greater (and more effective) computational power than the Turing machine. Specifically, each operation must be capable of processing multiple data pairs simultaneously. In other words, data must be placed as adjacent as possible, meaning the data placement mode must be non-linear. This is the motivation behind the proposal of the probe machine. The probe machine's data placement mode is spatially unconstrained, allowing any pair of data units to interact and process information directly.

9.2.2 Mathematical Model of the Probe Machine

This section presents the mathematical model of the probe machine, including the database, probe library, data controller, probe controller, probe operation, computing platform, detector, true solution storage, and residual support recycler.

The probe machine, denoted as PM, can be defined as a 9-tuple,

$$PM = (X, Y, \sigma_1, \sigma_2, \tau, \lambda, \eta, Q, C)$$

where, X represents the database, Y represents the probe library, σ_1 represents the data controller, σ_2 represents the probe controller, τ represents the probe operation, λ represents the computing platform, η represents the detector, Q represents the true solution storage, and C represents the residual support recycler. Next, we will define and explain these nine elements one by one.

9.2.2.1 Database X

The database of the probe machine defines a non-linear data placement mode. The database X consists of n data pools $X_1, X_2, \ldots, X_n$, that is, $X = \{X_1, X_2, \ldots, X_n\}$. For each $i \in \{1, 2, \ldots, n\}$, only one type of data x_i is stored in the data pool X_i, and x_i is massive, as shown in Fig. 9.1a and b. When only considering the type of data in the database X, X is regarded as a set of n elements, which can be represented as $X = \{x_1, x_2, \ldots, x_n\}$.

For $i \in \{1, 2, \ldots, n\}$, data x_i consists of two parts: one is called a data cell, and the other is called a data fiber. There is only one data cell, and there are p_i types of data fibers denoted as $x_i^1, x_i^2, \ldots, x_i^{p_i}$, where each type of data fiber x_i^l contains a massive number of identical copies in the data. The data cell is connected to the data fiber, and the same type of data fiber is distributed in the same connected area. Figure 9.1c shows a schematic diagram of the structure of data x_i: The small ball represents the data cell (as shown in Fig. 9.1d); a connected area of the small ball represents the same type of data fiber; p_i different color lines represent p_i types of data fibers $x_i^1, x_i^2, \ldots, x_i^{p_i}$; and $\Im(x_i)$ represents the set of all p_i types of data fibers in x_i, that is, $\Im(x_i) = \{x_i^1, x_i^2, \ldots, x_i^{p_i}\}$.

Assume that the database X has n different data types, denoted as $x_1, x_2, \ldots, x_n$. Let p_i represent the number of types of data fibers in x_i and use i to represent the

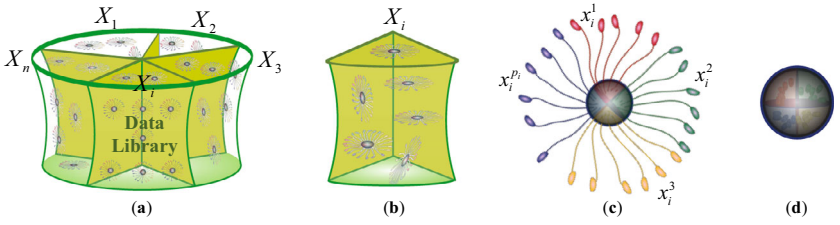


Fig. 9.1 Schematic diagram of the database [1]. (**a**) Database. (**b**) Data pool X_i. (**c**) Data x_i. (**d**) Data cell

data cell of x_i. Then, the data x_i can be represented as

$$x_i = \{i; x_i^1, x_i^2, \ldots, x_i^{p_i}\} \tag{9.1}$$

Assume that the data fibers belonging to different data are different, and let p represent the number of types of data fibers in the database. Then,

$$p = p_1 + p_2 + \ldots + P_n \tag{9.2}$$

Based on the above assumption, the following four points about the database X in the probe machine are emphasized:

1. The database X contains n types of data and p types of data fibers.
2. Each data pool X_i contains an abundant amount of data, denoted by x_i, meaning that this data is inexhaustible, where $i = 1, 2, \ldots, n$.
3. Each data pool x_i is equipped with a controllable output system, referred to as the *data controller*, which will be introduced later.
4. The database X can be represented in the form of the following matrix:

$$X = \begin{pmatrix} x_1^1 & x_2^1 & \ldots & x_n^1 \\ \vdots & & \ddots & \vdots \\ x_1^{p_1} & x_2^{p_2} & \ldots & x_n^{p_n} \end{pmatrix}$$

where the ith $(1 \leq i \leq n)$ column of X corresponds to all p_i types of data fibers in data x_i. When $i \neq t (1 \leq i, t \leq n)$, it is possible $p_i \neq p_t$, so X can be regarded as a matrix with a column number of n and unequal rows.

9.2.2.2 Probe Library

A probe is a device used to detect specific substances. The concept of a probe spans multiple fields, including biology, computer science, electronics, information security, and archaeology. Below, we provide a brief overview of its application in two key areas: biology and electronics.

In biology, probes are used to detect specific nucleic acid sequences. Typically, probes are short segments of single-stranded DNA or RNA, ranging from 20 to 500 base pairs in length. These probes are designed to be complementary to the target nucleic acid sequence. The process begins by heating double-stranded DNA to denature it into single strands or synthesizing single-stranded DNA molecules. The resulting single-stranded DNA can then be labeled using various methods, such as radioactive isotopes, fluorescent dyes, or enzymes. This labeled single-stranded DNA forms the probe. During the detection process, the probe hybridizes with the target sequence in the sample. If the probe is complementary to the target sequence, it will bind via hydrogen bonds. Any unhybridized excess probe is washed away, and methods like radioactive autoradiography, fluorescence microscopy, or enzyme-

linked amplification are employed to determine whether the target sequence is present and to locate its position within the sample.

In electronics, probes are used for various types of electronic testing, and they can be categorized based on their specific applications. One type is the printed circuit board (PCB) test probe, which tests circuit boards before component installation, primarily to detect issues such as short circuits. Another type is the online test probe, which inspects PCB circuit boards with installed components. Lastly, microelectronic test probes are typically used for wafer testing or integrated circuit (IC) chip detection.

Although similar to the existing concept of a probe, the probe discussed in this section is more abstract. Figuratively speaking, it functions like a "glue" that discovers and associates two data. Its precise definition is as follows: Let x_i^l and x_t^m represent two data fibers of data x_i. The probe between x_i^l and x_t^m is denoted as $\tau^{x_i^l x_t^m}$. This probe is an operator, commonly referred to as the probe operator, which must satisfy the following three conditions:

1. **Adjacency**: In the computing platform λ, $\tau^{x_i^l x_t^m}$ must be capable of accurately locating x_i^l and x_t^m.
2. **Uniqueness**: In the computing platform λ, $\tau^{x_i^l x_t^m}$ should only associate x_i^l with x_t^m, without involving any other data fibers.
3. **Probability**: In the computing platform λ, $\tau^{x_i^l x_t^m}$ must be able to perform a specific attribute probe operation (such as a connect operation, transmission operation, etc.) while identifying x_i^l and x_t^m. The result of the operation is recorded as $\tau^{x_i^l x_t^m}(x_i, x_t)$.

If there exists a probe $\tau^{x_i^l x_t^m}$ between two data fibers x_i^l and x_t^m in the database, then x_i^l and x_t^m are considered **probable**, and x_i^l and x_t^m are referred to as the **probe objects** of the probe $\tau^{x_i^l x_t^m}$. Consequently, the data x_i and x_t are also considered probable. Conversely, if no probe exists between x_i^l and x_t^m, then x_i^l and x_t^m are **non-probable**. Furthermore, if no probe exists between any pair of data fibers from x_i and x_t, then x_i and x_t are **non-probable**.

Let $X' \subseteq X$ represent any subset of the database X. We denote the set of probe operators corresponding to all probable data pairs in X' as $\tau(X')$, and refer to it as the **probe subset** of X'. Below, we introduce two types of probe operators in the probe machine: **connection-type probe operators** (connection operators) and **transmission-type probe operators** (transmission operators).

(1) Connected Operator

The connection-type probe operator for data fibers x_i^l and x_t^m, commonly referred to as the **connection-operator**, is the probe operator $\tau^{x_i^l x_t^m}$ that establishes a connection between the target data fibers x_i^l and x_t^m. To distinguish different types of probe operators, we use $\overline{x_i^l x_t^m}$ to represent the connection-operator that links x_i^l and x_t^m. It is important to note that the connection-operator $\overline{x_i^l x_t^m}$ and $\overline{x_t^m x_i^l}$ refer to the same probe operator, meaning the connection-operator does not distinguish the direction of the connection between the two data fibers. In the computing

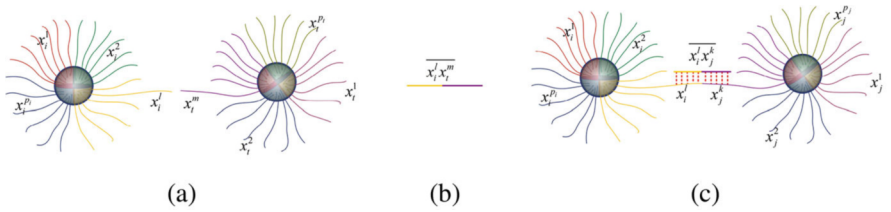


Fig. 9.2 Schematic diagram of the connection operator action process [1]. (**a**) Two data. (**b**) Connection probe. (**c**) Probe operation

platform λ, the connection operator can identify and establish a connection between the target data fibers. This concept is illustrated in Fig. 9.2. In Fig. 9.2a, we show a schematic diagram of the structure of data x_i and x_t. Figure 9.2b depicts the connection-operator $\overline{x_i^l x_t^m}$, which can be interpreted as a complementary sequence formed by half of the data fibers taken from x_i^l and x_t^m (specifically, the half not connected to the data cell). This complementary sequence can adsorb and connect the target data fibers, thereby linking the two data fibers x_i^l and x_t^m together and, consequently, connecting the data x_i and x_t. More precisely, the connection of data x_i and x_t is achieved through the connection of data fibers x_i^l and x_t^m. The symbol $\overline{x_i^l x_t^m}(x_i, x_t) \triangleq x_i x_t^{\overline{x_i^l x_t^m}}$ represents the result of the connected operator's action, as shown in Fig. 9.2c.

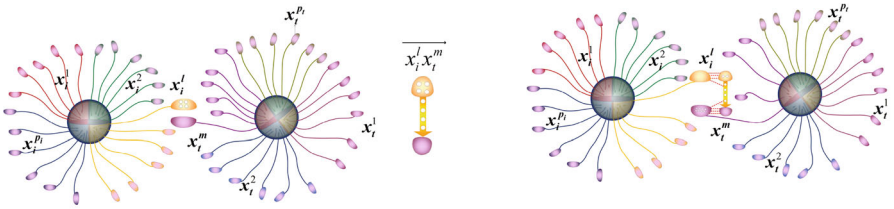
We refer to the data fibers capable of processing information through the connection-operator as **connection-type data fibers**, and the corresponding data as **connection-type data**. Typically, connection-type data refers to data where all the data fibers are connection-type data fibers. If every data in the database is connection-type data, we call this database a **connection-type database**.

In Fig. 9.2, we can abstractly consider x_i^l and x_t^m as two DNA strands embedded in nanoparticles, where data cells are represented by nanoparticles embedded in data fibers. Then, the connection-operator $\overline{x_i^l x_t^m}$ can be seen as formed by the complementary strands of half of the DNA strands representing data fibers x_i^l and x_t^m, connected by hydrogen bond attraction.

The connection operator is a formal mathematical abstraction derived from several real-world instances, with the biological probe as an example. From the definition of the connection operator, its primary function is to locate and connect two target data fibers. Therefore, the connection operator does not consider the direction of the connection. In contrast, the transmission operator described below has a specific direction.

(2) Transmission Operator

Assume that all data fibers on data x_i and x_t carry information, as shown in Fig. 9.3a. The transmission-type probe operator for data fibers x_i^l and x_t^m, also known as the **transmission-operator**, is the probe operator $\tau^{x_i^l x_t^m}$ that transmissions the information from the source data fiber x_i^l to the destination data fiber x_t^m. This



(a) (b) (c)

Fig. 9.3 Schematic diagram of the transmission operator action process [1]. (**a**) Two data. (**b**) Transmission probe. (**c**) Probe operation

operator is represented by the symbol $\overrightarrow{x_i^l x_t^m}$. Unlike the connection operator, the transmission operator distinguishes the connection direction between the two data fibers. Specifically, $\overrightarrow{x_i^l x_t^m}$ and $\overrightarrow{x_t^m x_i^l}$ represent different transmission operators. In the computing platform λ, the transmission operator can identify the target data fibers (source and destination) and transmission the information from x_i^l to x_t^m by establishing a connection between them. This process is illustrated in Fig. 9.3. Figure 9.3a shows the structure of data x_i and x_t, while Fig. 9.3b depicts the transmission operator for x_i^l and x_t^m. Figure 9.3c illustrates the result obtained after applying the transmission operator. The process can be described as follows:

1. The transmission operator $\overrightarrow{x_i^l x_t^m}$ locates and connects the data fibers x_i^l (source data fiber) and x_t^m (destination data fiber).
2. It then performs operations on the source data fiber x_i^l, transmitting the information it contains to the destination data fiber x_t^m.

We use the symbol $\overrightarrow{x_i^l x_t^m}(x_i, x_t) \triangleq x_i \overrightarrow{x_t^{x_i^l x_t^m}}$ to represent the result after the transmission operator $\overrightarrow{x_i^l x_t^m}$ has been applied.

Data fibers processing information through the transmission operator are called **transmission data fibers**, and the corresponding data are called **transmission data**. Usually, transmission data refers to the data that contains only transmission data fibers. If every data in the database is transmission data, then this database is called a **transmission database**.

(3) Probe Library Construction

Based on the database, a probe library can be constructed, which satisfies the following two principles:

1. No probe exists between any pair of data fibers x_i^ℓ and x_i^t on the same data x_i, where $1 \leq i \leq n$ in the database $X = \{x_1, x_2, \ldots, x_n\}$.
2. A probe may exist between any pair of data fibers x_i^ℓ and x_t^m, where x_i^ℓ is from data x_i and x_t^m is from data x_t, and $i \neq t$.

Based on the above two principles, the probe libraries constructed on the connection-type and transmission-type databases are referred to as the connection-type and transmission-type probe libraries, respectively. The connection-type and transmission-type probe libraries store connection operators and transmission operators, respectively. The following proposition holds.

Proposition 9.1 *Let $X = \{x_1, x_2, \ldots, x_n\}$ be the database of the probe machine, where each data x_i contains p_i types of data fibers, $i = 1, 2, \ldots, n$, If X is a connection-type database, then its corresponding connection-type probe library contains at most $\frac{1}{2}\{p^2 - (p_1^2 + p_2^2 + \ldots + p_n^2)\}$ different connection operators; if X is a transmission-type database, then its corresponding transmission-type probe library contains at most $p^2 - (p_1^2 + p_2^2 + \ldots + p_n^2)$ different transmission operators.*

Proof For the connection-type database, if each data fiber is represented as a vertex, an edge is linked between two distinct vertices if and only if there is a connection operator between the data fibers represented by these two vertices. According to the first property of the probe library, the graph G constructed using this method is an n-partite undirected simple graph, where the n parts correspond exactly to the n data elements $x_1, x_2, \ldots, x_n$, and the number of vertices in the ith part is denoted by p_i for $i = 1, 2, \ldots, n$. Thus, the number of edges in G corresponds to the number of distinct connection operators in the connection-type probe library. Therefore, based on the second property of the probe library, when G is a complete n-partite graph, the number of edges (or the number of connection operators) is maximized and is given by:

$$\frac{1}{2}\left\{p^2 - \left(p_1^2 + p_2^2 + \ldots + p_n^2\right)\right\}$$

Similarly, for the transfer-type database, if each data fiber is represented as a vertex, a directed edge is linked between two distinct vertices (data fibers) x_i^l and x_t^m if and only if there is a transfer operator $\overrightarrow{x_i^l x_t^m}$, where x_i^l is the source data fiber and x_t^m is the destination data fiber. Based on the first property of the probe library, the graph constructed using this method is a directed n-partite graph G, where the n parts correspond exactly to the n data elements $x_1, x_2, \ldots, x_n$, with the number of vertices in the ith part denoted by p_i for $i = 1, 2, \ldots, n$. The number of directed edges in G corresponds to the number of distinct transfer operators in the transfer-type probe library. Therefore, based on the second property of the probe library, when G is a complete n-partite directed graph (i.e., there is a directed edge between every ordered pair of vertices), the number of directed edges (or the number of transfer operators) is maximized, and is given by:

$$p^2 - \left(p_1^2 + p_2^2 + \ldots + p_n^2\right)$$

□

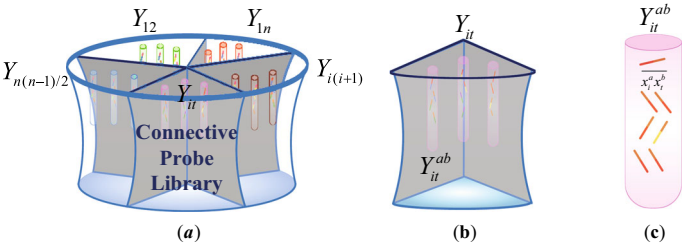


Fig. 9.4 Schematic diagram of the connection-type probe library [1]. (**a**) Connection probe library. (**b**) Sub-probe library. (**c**) Probe pool

According to the construction principle of the probe library, the structure of the probe library can be composed of sub-probe libraries. Based on the database $X = \{x_1, x_2, \ldots, x_n\}$, the set of all possible probes between data x_i and x_t is called an (i, t)**-complete sub-probe library**, denoted as Y_{it}, where $i, t = 1, 2, \ldots, n, i \neq t$. When not distinguishing specific data, Y_{it} is called the **complete sub-probe library**. All complete sub-probe libraries constitute the probe library Y, that is

$$Y = \bigcup_{i,j=1,2,\ldots,n,i\neq j} Y_{ij}$$

Based on the previous definition, it is evident that for the connection-type probe library, $Y_{it} = Y_{ti}$. Therefore, the connection-type probe library Y consists of $\frac{n(n-1)}{2}$ complete sub-probe libraries, as illustrated in Fig. 9.4a. Figure 9.4b shows the schematic diagram of the structure of the complete sub-probe library Y_{it}. Since data x_i and x_t contain p_i and p_t types of data fibers, respectively, Y_{it} contains a total of $p_i \times p_t$ probes, that is, $|Y_{it}| = p_i \times p_t$. The following formula gives the elements contained in Y_{it}:

$$\begin{aligned} Y_{it} = \{&\overline{x_i^1 x_t^1}, \overline{x_i^1 x_t^2}, \ldots, \overline{x_i^1 x_t^{pt}}; \overline{x_i^2 x_t^1}, \overline{x_i^2 x_t^2}, \ldots, x_i^2 x_t^{p_t}; \\ &\ldots; x_i^{p_i} x_t^1, x_i^{p_i} x_t^2, \ldots, x_i^{p_i} x_t^{p_t}\} \end{aligned} \tag{9.3}$$

For each $a = 1, 2, \ldots, p_i$ and $b = 1, 2, \ldots, p_t$, a probe pool is constructed to store the probe operator $\overline{x_i^a x_t^b}$ in Y_{it}, and we denote this probe pool by Y_{it}^{ab} (as shown in Fig. 9.4c). Note that Y_{it}^{ab} only stores the probe operator of this type, $\overline{x_i^a x_t^b}$, and the amount stored is sufficient.

For the transmission-type probe library, since $Y_{it} \neq Y_{ti}$, there are a total of $n(n-1)$ types of complete sub-probe libraries, which is exactly twice the number of types of complete sub-probe libraries in the connection-type probe library, as illustrated in Fig. 9.5a. Additionally, the types of probe operators contained in the complete sub-

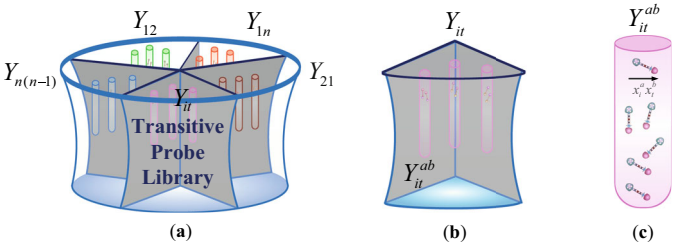


Fig. 9.5 Schematic diagram of the transmission type probe library structure [1]. (**a**) Transmission probe library. (**b**) Sub-probe library. (**c**) Probe pool

probe library in the transmission-type probe library are exactly twice the number of probe operators contained in the complete sub-probe library in the connection-type probe library. Figure 9.5b shows the schematic diagram of the structure of the transmission-type complete sub-probe library Y_{it}, which contains a total of $2p_i\,p_t$ probes, as given by the following formula:

$$Y_{it} = \{\overrightarrow{x_i^1 x_t^1}, \overrightarrow{x_t^1 x_i^1}, \ldots, \overrightarrow{x_i^1 x_t^{p_t}}, \overrightarrow{x_t^{p_t} x_i^1}; \ldots; \overrightarrow{x_i^{p_i} x_t^1}, \overrightarrow{x_t^1 x_i^{p_i}}, \ldots, \overrightarrow{x_i^{p_i} x_t^{p_t}}, \overrightarrow{x_t^{p_t} x_i^{p_i}}\} \tag{9.4}$$

Similarly, for each $a = 1, 2, \ldots, p_i$ and $b = 1, 2, \ldots, p_t$, a probe pool is constructed to store the probe operator $\overrightarrow{x_i^a x_t^b}$ in Y_{it}, and we denote this probe pool by Y_{it}^{ab} (as shown in Fig. 9.5c). It is important to note that Y_{it}^{ab} stores only the probe operator of this specific type, $\overrightarrow{x_i^a x_t^b}$, and the amount stored is sufficient.

9.2.2.3 Data Controller σ_1 and Probe Controller σ_2

The data controller, denoted by σ_1, is responsible for extracting the required amount of data from the data pool and transmitting it to the computing platform λ. Each data pool is equipped with its own controller, meaning that the number of data controllers in the database equals the number of data pools, with a total of n data controllers.

The probe controller, denoted by σ_2, is responsible for extracting the required number of probes from the probe pool and sending them to the computing platform λ. Each probe pool is equipped with a controller, so the number of probe controllers in the probe library corresponds to the number of probe pools. Therefore, the connection-type probe library contains a total of $\frac{p^2-(p_1^2+p_2^2+\ldots+p_n^2)}{2}$ probe controllers, while the transmission-type probe library contains a total of $p^2 - (p_1^2 + p_2^2 + \ldots + p_n^2)$ probe controllers.

9.2.2.4 Probe Operation τ

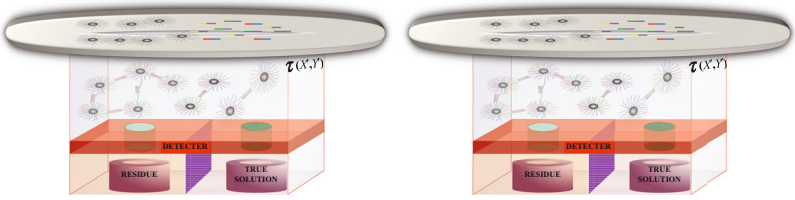
The probe operation is used to identify and connect target data fibers. Specifically, for two probeable data elements, x_i and x_t, in the database, let x_i^l and x_t^m be the data fibers on x_i and x_t, respectively. The **basic probe operation** consists of two stages: the preparation stage and the execution stage. In the preparation stage, under the control of the data controller and the probe controller, both the data elements x_i and x_t, along with the probe operator $\tau^{x_i^l x_t^m}$, are placed into the computing platform λ, allowing the operator $\tau^{x_i^l x_t^m}$ to identify and locate x_i^l and x_t^m. In the execution stage, the probe operator $\tau^{x_i^l x_t^m}$ acts on x_i^l and x_t^m to perform the required operations.

The probe operation is a process in which multiple basic probe operations are executed simultaneously. Its formal definition can be described as follows: Let X' be a subset of the database X, and Y' be a subset of the probe library $\tau(X')$. Based on the probe operation on X' and Y', represented by the symbol τ, the result is obtained after the probe subset Y' acts on the data subset X'. In other words, each probe operator in Y' acts on the corresponding data pairs in X'. All probe operators in Y' simultaneously perform their respective basic probe operations on the corresponding data elements in X'. The result of this operation is referred to as the solution of τ, denoted by Θ. That is:

$$\tau(X', Y') = \Theta \tag{9.5}$$

What is Θ? This question is intricately linked to the computing platform λ. To address this, the concept of the probe operation graph is introduced. The probe operation graph associated with a data subset X' and a probe subset Y', denoted as $G^{(X',Y')}$, refers to the **topological structure of the true solution aggregate** after the probe operation. The vertex set $V(G^{(X',Y')})$ represents the data in the true solution aggregate, while the edge set $E(G^{(X',Y')})$ represents the probes in the true solution aggregate. For specific problems, the probe operation graph can be determined, and its primary function is to identify the solution. For example, when solving the Hamiltonian cycle problem in subsequent sections, the probe operation graph may be either a 4-cycle or a 5-cycle. When solving the vertex coloring problem of a graph G, the probe operation graph corresponds to the graph G itself. In the case of the maximum clique problem for a given graph G, the probe operation graph is the topological structure corresponding to the largest clique.

In fact, the probe operation can be viewed as a reaction process: the object of the reaction is the data subset X', the executor of the reaction is the probe operator in Y', and the carrier of the reaction is the computing platform λ. Next, we will provide a detailed introduction to the structure and properties of the computing platform λ.



(a) (b)

Fig. 9.6 Schematic diagram of the computing platform, detector, true solution storage, and residual branch collector [1]. (**a**) Connection type. (**b**) Transmission type

9.2.2.5 Computing Platform λ

The computing platform, denoted as λ, serves as a specialized environment for performing probe operations. Its primary function is to assist the probe operator in quickly and accurately locating the target data fibers and executing the corresponding basic probe operations. The structure of the computing platform is depicted in Fig. 9.6, where Fig. 9.6a illustrates the connection-type computing platform, and Fig. 9.6b shows the transmission-type computing platform. We define an aggregate that connects two data elements via a corresponding probe as a 2-data aggregate. Similarly, if a 2-data aggregate can be further combined with another data element through a probe operator, the resulting entity is referred to as a 3-data aggregate. By extension, if an aggregate is formed through several probe operations based on $m \geq 2$ data elements, it is called an m-data aggregate, where m denotes the order of the aggregate. In particular, each individual data element x_i, for $i = 1, 2, \ldots, n$, is termed a 1-data aggregate. Aggregates are typically represented by the symbol M or with a subscript for more specific cases, and the order of an aggregate is denoted by $|M|$.

The computing platform λ has three fundamental functions.

Function 1: High Aggregation When the probe $\tau^{x_i^l x_t^m}$ is placed into the computing platform λ, the platform seeks two target data fibers, x_i^l and x_t^m, that can generate higher-order data aggregates. Specifically, the following principles are adhered to:

1. Let M_1, M_2, and M_3 be three data aggregates in the computing platform λ, where M_1 contains the data fiber x_i^l, and both M_2 and M_3 contain the data fiber x_t^m. If x_i^l and x_t^m are probeable, and if $|M_2| > |M_3|$, the platform will guide $\tau^{x_i^l x_t^m}$ to select x_i^l in M_1 and x_t^m in M_2 for the basic probe operation.
2. Suppose M_1 and M_2 are two data aggregates in the computing platform λ, where both M_1 and M_2 contain data x_i and x_t. If the probe operator $\tau^{x_i^l x_t^m}$ can only perform basic probe operations on M_1, M_2, or between M_1 and M_2, then the following conditions apply: First, when $|M_1| > |M_2|$, the computing platform λ will guide $\tau^{x_i^l x_t^m}$ to select two data fibers x_i^l from M_1 and x_t^m from M_2 and perform basic probe operations; second, when $|M_1| = |M_2|$, the computing

platform λ will guide $\tau^{x_i^l x_t^m}$ to select the data aggregate containing the most probe operators, and then select two data fibers, x_i^l and x_t^m, from this aggregate to perform basic probe operations; third, when $|M_1| = |M_2|$ and both M_1 and M_2 contain the same number of probe operators, the computing platform λ will randomly select two data fibers from either of M_1 and M_2 to perform the basic probe operation. It should be noted that the data aggregate cannot grow indefinitely. It is constrained by a specific threshold value, which is another fundamental function of the computing platform λ.

Function 2: Threshold In the computing platform λ, the order of the data aggregate after the probe operation must be exactly equal to the order of the probe operation graph $G^{(X',Y')}$. The number of basic probe operations must exactly match the number of edges in $G^{(X',Y')}$. Therefore, if the sum of the orders of the two aggregates exceeds the order of $G^{(X',Y')}$, even if there is probable data, the computing platform λ will prevent them from performing basic probe operations.

Function 3: Uniqueness In the computing platform λ, any data aggregate M with an order greater than 1 contains at most one instance of the same data type. Furthermore, during the formation of M, at most one basic probe operation is performed between any pair of data elements.

9.2.2.6 Detector η

The probe machine utilizes the probe operation graph to detect solutions (see Sect. 9.2.2.4). For a given problem, after performing the probe operation, many aggregates (also referred to as solutions to the problem) are generated on the computing platform λ. The necessary and sufficient condition for an aggregate to be considered a true solution to the problem is that its topological structure is isomorphic to the probe operation graph. Aggregates that are not isomorphic to the probe operation graph are termed residual aggregates (or residuals). The task of the detector η is to identify the true solution on the computing platform λ and distinguish it from the residuals. Specifically, let X' and Y' represent the required subsets of data and probes for the problem, respectively, and let $G^{(X',Y')}$ denote the corresponding probe operation graph after the probe operation. The basic function of the probe detector η is as follows:

Let M be any data aggregate in the computing platform λ. If the order of M does not match the number of vertices in $G^{(X',Y')}$, or the number of probe operators in M is not equal to the number of edges in $G^{(X',Y')}$, then M is considered a residual aggregate and is separated into the residual recycler (as shown in Fig. 9.6). However, if the order of M is equal to the number of vertices in $G^{(X',Y')}$, and the number of probes in M is equal to the number of edges in $G^{(X',Y')}$, then each probe operator uniquely corresponds to a pair of probable data fibers. In this case, M is the true solution to the problem and is separated into the true solution storage (as shown in Fig. 9.6).

9.2.2.7 Steps of Probe Operation

Let X' and Y' represent the subsets of data and probes required for a given problem. Based on the previous discussion, the specific steps of the probe operation can be described as follows:

Step 1: Determine the probe operation graph $G^{(X',Y')}$. Based on this graph, we can estimate the probability of generating $G^{(X',Y')}$ during the probe operation. Due to the computing platform's high aggregation, threshold, and uniqueness properties, this probability is expected to be significantly high.
Step 2: Determine the quantity of each type of data in the data sublibrary X' and the quantity of each type of probe in the sub-probe library Y'. The method is as follows: based on the probability of $G^{(X',Y')}$ being generated, and under the assumption that a true solution exists in Θ after the probe operation, determine the quantity of each type of data and each type of probe.
Step 3: Using the data controller σ_1 and the probe controller σ_2, extract the determined quantities of each type of data in X' and each type of probe in Y' from the corresponding data and probe pools. These are then placed into the computing platform λ.
Step 4: Perform the probe operation $\tau(X', Y')$ on the computing platform λ, activating the detector η. The true solution is separated into the true solution storage, and the residual aggregates are placed into the residual recycler. The topological structure of the true solution obtained in the above steps must be isomorphic to $G^{(X',Y')}$. However, different true solutions may have different weights (corresponding to different data fibers).

9.2.2.8 True Solution Storage and Residual Recycler

The true solution memory plays a crucial role in the computing platform. Its primary function is to securely store the true solutions and enable accurate retrieval of these solutions. During the probe operation process, the computing platform λ may generate aggregates that are not isomorphic to the probe operation graph $G^{(X',Y')}$, referred to as residues. These residues are then directed to the residue recycler, as depicted in Fig. 9.6. The main task of the residue recycler is to collect the residues produced by the computing platform after the probe operation, perform fine separation and processing, and subsequently reintegrate the data into the corresponding database.

9.2.2.9 Structure Model of Probe Machine

Based on the previous discussion of the nine fundamental elements of the probe machine, two overall structural models can be identified: one based on a connection type and the other on a transmission type, as illustrated in Fig. 9.7.

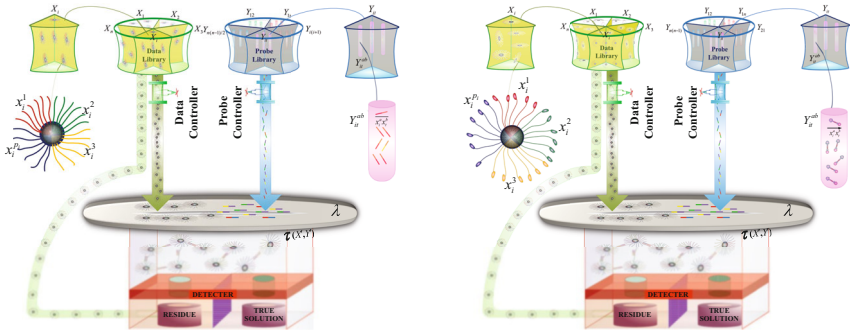


Fig. 9.7 Schematic diagram of the probe machine structure model [1]. (**a**) Connection-type probe machine model. (**b**) Transfer-type probe machine model

To further elucidate the computational nature of the probe machine, the following section will explore the solution method for the Hamiltonian problem. Subsequently, an implementation method for the connection-type probe machine, based on DNA nanotechnology, will be presented, along with its application to the graph coloring problem.

9.3 Probe Machine Solves Hamilton Circle Problem

The Hamiltonian problem involves determining whether a given graph contains a Hamiltonian cycle, i.e., a cycle that passes through each vertex of the graph exactly once. This problem, first proposed by Hamilton [22] in 1856, is a well-known **NP**-complete problem. Below, we present a connection-type probe machine model for solving the Hamiltonian problem.

Let G be a simple undirected graph with vertex set $V(G)$ and edge set $E(G)$, where $V(G) = \{v_1, v_2, \ldots, v_n\}$. For any vertex $v_k \in V(G)$, let $E(v_k)$ denote the set of edges associated with vertex v_k, i.e., $E(v_k) = \{v_k v_i \mid v_k v_i \in E(G), 1 \leq i \leq n\}$, and let $E^2(v_k)$ represent the set of 2-length paths centered on v_k (as shown in Fig. 9.8a), that is,

$$E^2(v_k) = \{v_i v_k v_j \stackrel{\triangle}{=} x_{kij};\ v_i, v_j \in N(v_k);\ i \neq j\} \tag{9.6}$$

where $N(v_k) = \{x \mid x v_k \in E(G)\}$. Based on $E^2(v_k)$, we construct the connection-type probe database for the Hamilton problem X as follows:

$$X = \cup_{k=1}^{n} E^2(v_k) = \cup_{k=1}^{n} \{x_{kij} | v_i, v_j \in N(v_k);\ i \neq j\} \tag{9.7}$$

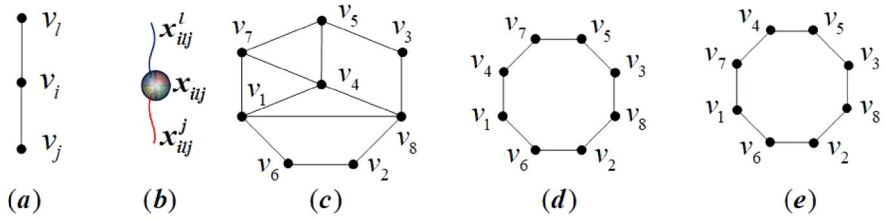


Fig. 9.8 Probe machine model for solving the Hamilton problem [1]

where each data x_{kij} has exactly two data fibers, denoted as x^i_{kij}, x^j_{kij}, as shown in Fig. 9.8b.

Based on the database X constructed previously, we now build the probe library Y. It is generally assumed that the order of the graph G is greater than or equal to 5. We now analyze two cases for two vertices v_i and v_t.

Case 1: v_i and v_t are not adjacent. In this case, the two data x_{ilj} and x_{tab} in the database X contain probes if and only if the following condition holds:

$$|\{i, l, j\} \cap \{t, a, b\}| = |\{l, j\} \cap \{a, b\}| = 1 \tag{9.8}$$

This condition implies that $i \notin \{t, a, b\}$, $t \notin \{l, j\}$, and the intersection $\{l, j\} \cap \{a, b\}$ contains exactly one element.

Case 2: v_i is adjacent to v_t. In this case, a probe between data x_{ilj} and x_{tab} exists if and only if one of the following conditions holds:

$$|\{i, l, j\} \cap \{t, a, b\}| = |\{j, l\} \cap \{a, b\}| = 1;$$

or

$$t \in \{l, j\}, i \in \{a, b\} \text{ and } |\{l, j\} \cap \{a, b\}| = 0.$$

When solving the Hamiltonian problem using the probe machine model, it is unnecessary to take the set of 2-length paths centered at each vertex v_k as a subset of the database. Instead, we can select a vertex cover in the graph G and then construct the database for the 2-length paths with centers in the vertex cover. Naturally, the minimum vertex cover set provides the optimal choice. Next, we outline the steps for solving the Hamiltonian problem using the connection-type probe machine model. We illustrate these steps with an example based on the 8-order graph shown in Fig. 9.8c.

Step 1. Build the Database

One can readily check that $\{v_1, v_2, v_3, v_4, v_5\}$ is a minimum vertex cover of the graph. So, the database is defined as

$$X = E^2(v_1) \cup E^2(v_2) \cup E^2(v_3) \cup E^2(v_4) \cup E^2(v_5)$$

where $E^2(v_1) = \{x_{174}, x_{178}, x_{176}, x_{148}, x_{146}, x_{186}\}$, $E^2(v_2) = \{x_{268}\}$, $E^2(v_3) = \{x_{358}\}$, $E^2(v_4) = \{x_{458}, x_{451}, x_{457}, x_{481}, x_{487}, x_{417}\}$, $E^2(v_5) = \{x_{534}, x_{537}, x_{547}\}$. There are 17 types of data in the database X and 34 types of data fibers. The data fibers on each type of data are as follows:

$\Im(x_{174}) = \{x^7_{174}, x^4_{174}\}$,$\Im(x_{178}) = \{x^7_{178}, x^8_{178}\}$,$\Im(x_{176}) = \{x^7_{176}, x^6_{176}\}$,$\Im(x_{148}) = \{x^4_{148}, x^8_{148}\}$,
$\Im(x_{146}) = \{x^4_{146}, x^6_{146}\}$,$\Im(x_{186}) = \{x^8_{186}, x^6_{186}\}$,$\Im(x_{268}) = \{x^6_{268}, x^8_{268}\}$,$\Im(x_{358}) = \{x^5_{358}, x^8_{358}\}$,
$\Im(x_{458}) = \{x^5_{458}, x^8_{458}\}$,$\Im(x_{451}) = \{x^5_{451}, x^1_{451}\}$,$\Im(x_{457}) = \{x^5_{457}, x^7_{457}\}$,$\Im(x_{481}) = \{x^1_{481}, x^8_{481}\}$,
$\Im(x_{487}) = \{x^7_{487}, x^8_{487}\}$,$\Im(x_{417}) = \{x^7_{417}, x^1_{417}\}$,$\Im(x_{534}) = \{x^3_{534}, x^4_{534}\}$,$\Im(x_{537}) = \{x^3_{537}, x^7_{537}\}$,
$\Im(x_{547}) = \{x^4_{547}, x^7_{547}\}$.

Step 2. Build the Probe Library

According to the two situations of adjacency and non-adjacency, we construct a sub-probe library based on 34 types of data fibers.

$$Y_{12} = \overline{x^8_{178}x^8_{268}}, \overline{x^6_{176}x^6_{268}}, \overline{x^8_{148}x^8_{268}}, \overline{x^6_{146}x^6_{268}},$$

$$Y_{13} = \{\overline{x^8_{178}x^8_{358}}, \overline{x^8_{148}x^8_{358}}, \overline{x^8_{186}x^8_{358}}\},$$

$$Y_{14} = \{\overline{x^7_{178}x^7_{457}}, \overline{x^8_{178}x^8_{458}}, \overline{x^7_{176}x^7_{457}}, \overline{x^7_{176}x^7_{487}}, \overline{x^8_{186}x^8_{458}}, \overline{x^8_{186}x^8_{487}}, \overline{x^4_{174}x^1_{451}},$$

$$\overline{x^4_{174}x^1_{481}}, \overline{x^4_{148}x^1_{451}}, \overline{x^4_{148}x^1_{417}}, \overline{x^4_{146}x^1_{451}}, \overline{x^4_{146}x^1_{481}}, \overline{x^4_{146}x^1_{417}}\},$$

$$Y_{15} = \{\overline{x^7_{174}x^7_{537}}, \overline{x^4_{174}x^4_{534}}, \overline{x^7_{178}x^7_{537}}, \overline{x^7_{178}x^7_{547}}, \overline{x^7_{176}x^7_{537}}, \overline{x^7_{176}x^7_{547}}, \overline{x^4_{148}x^4_{534}},$$

$$\overline{x^4_{148}x^4_{547}}, \overline{x^4_{146}x^4_{534}}, \overline{x^4_{146}x^4_{547}}\},$$

$$Y_{23} = \{\overline{x^8_{268}x^8_{358}}\}$$

$$Y_{24} = \{\overline{x^8_{268}x^8_{458}}, \overline{x^8_{268}x^8_{481}}, \overline{x^8_{268}x^8_{487}}\}$$

$$Y_{34} = \{\overline{x^5_{358}x^5_{451}}, \overline{x^5_{358}x^5_{457}}, \overline{x^8_{358}x^8_{481}}, \overline{x^8_{358}x^8_{487}}\}$$

$$Y_{35} = \{\overline{x_{358}^{5} x_{534}^{3}}, \overline{x_{358}^{5} x_{537}^{3}}\}$$

$$Y_{45} = \{\overline{x_{487}^{7} x_{537}^{7}}, \overline{x_{417}^{7} x_{537}^{7}}, \overline{x_{458}^{5} x_{534}^{4}}, \overline{x_{458}^{5} x_{547}^{4}}, \overline{x_{451}^{5} x_{534}^{4}}, \overline{x_{451}^{5} x_{547}^{4}}, \overline{x_{457}^{5} x_{543}^{4}}\}$$

Step 3. Execute Probe Operation
Using the data controller σ_1, a suitable amount of data $x_{174}, x_{178}, x_{176}, x_{148}, x_{146}, x_{186}, x_{268}, x_{358}, x_{458}, x_{451}, x_{457}, x_{481}, x_{487}, x_{417}, x_{534}, x_{537}, x_{547}$ are extracted from the database X and placed into the computing platform λ. Using the probe controller σ_2, a suitable amount of probes from the sub-probe libraries $Y_{12}, Y_{13}, Y_{14}, Y_{15}, Y_{23}, Y_{24}, Y_{34}, Y_{45}$ are extracted and placed into the computing platform λ. The probe operation τ is then performed in λ, and under the functionality of λ and τ, the solution to the problem is obtained. Finally, the detector η checks the solution.

Step 4. Check the Solution
The detector η places 4-order or 5-order aggregates into the true solution memory and directs the remaining aggregates to the residual recovery. As illustrated in Fig. 9.8d and e, the two 8-order cycles represent the Hamiltonian cycles identified during the process.

9.4 A Technology for Implementing a Connected Probe Machine

As previously discussed, a computer is a machine developed based on a specific computational model, using materials capable of executing this model. The probe machine described in this chapter represents one such computational model. The challenge now is to identify suitable materials for constructing a computer based on the probe machine model. This is a complex problem. In this section, we present a model for implementing a connection-type probe machine called the nano-DNA probe machine model. In this model, the primary material for data is a composite of nanoparticles and DNA molecules, while the primary material for probes is DNA molecules. Although current nanotechnology and biotechnology, especially detection technology, may struggle to solve large-scale practical problems [23], advancements in detection technology could make the development of practical nano-DNA probe computers possible.

Regarding the transmission probe machine model, a conjecture is proposed in the literature [1], suggesting that the data in this model is a composite, with the information carried by the data fibers akin to neurotransmitters (such as acetylcholine). The probe operation in this model is hypothesized to be executed via "action potentials," similar to those in biological nervous systems. Research in this area is still exploratory, and further discussions on the transmission-type probe machine model and biological neural networks will be presented in the next section.

Here, we focus primarily on the nano-DNA probe machine model, which includes the database, the probe library, and the detector.

Database In the nano-DNA probe machine model, the data design within the database is as follows: nanoparticles are used as data cells, and DNA strands serve as data fibers, as shown in Fig. 9.1c and d. This data is vividly referred to as "little stars." A data element x_i carrying p_i types of data fibers can be realized by the following method: first, the nanoparticles are evenly divided into p_i connected regions, and then a sufficient quantity of the same type of DNA strands (i.e., data fibers) are connected in each region. Refer to [24] to the technique of connecting DNA strands to nanoparticles.

Probe Library The design of the probes within the probe library of the nano-DNA probe machine model is as follows: Suppose x_i^a and x_t^b represent two fibers on data x_i and x_t, respectively. If these two fibers are capable of being probed, the probe is denoted as $\overline{x_i^a x_t^b}$, which is a complementary strand of DNA formed by half of each fiber from x_i^a and x_t^b, as shown in Fig. 9.2.

Detector Designing a detector for the nano-DNA probe machine model is challenging with current technology. Biochemical experiments typically use PCR instruments or electron microscopes for detection. However, this method is still immature. Once detection technology advances, the nano-DNA probe computer will become a practical solution.

Next, we will explain the computation process of the nano-DNA probe machine model in detail, using a specific graph vertex coloring example. This model's core principle of probe design is to ensure that adjacent vertices are assigned different color probes.

For a k-colorable graph, if a vertex v consistently belongs to the same color class under any k-coloring (in terms of color isomorphism), then v is referred to as the coloring fixed vertex of the graph. Consider the 4-colorable maximal planar graph shown in Fig. 9.9, where a maximal planar graph is a planar graph whose faces are triangles. Its vertex set is $\{1, 2, \ldots, 12\}$. Let V' represent the set of coloring fixed vertices of this graph. It is easy to verify that $V'' = \{1, 2, 3, 4, 5\}$. The method for

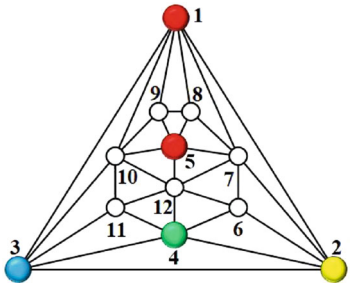


Fig. 9.9 A 12-order 4-color maximal planar graph and its color invariant set [1]

Table 9.1 Possible colorings of each vertex in the graph shown in Fig. 9.10, a total of 21

1	2	3	4	5	6	7	8	9	10	11	12
r_1				r_5	r_6					r_{11}	
	y_2						y_8	y_9	y_{10}	y_{11}	y_{12}
		b_3			b_6	b_7	b_8	b_9			b_{12}
			g_4			g_7	g_8	g_9	g_{10}		

solving all colorings of this graph using the nano-DNA probe machine model is provided below.

Based on the set V'', we construct a color table with the fewest colors possible, as shown in Table 9.1. The numbers at the top, $1, 2, \ldots, 12$, represent the graph's vertices. Each vertex i ($1 \leq i \leq 12$) is located in a column that indicates the allowed color for that vertex. The color marks r, y, b, and g represent red, yellow, blue, and green, respectively, with subscripts corresponding to the vertex identifier. For example, r_6 indicates that vertex 6 is colored red. It is important to note that because vertices 1, 2, 3, 4, and 5 are coloring fixed vertices, their colors are fixed without losing generality as red, yellow, blue, green, and red, respectively.

(1) Database Construction Based on Table 9.1, the database consists of 12 data elements, denoted as $x_1, x_2, \ldots, x_{12}$. For each data element x_i ($i = 1, 2, \ldots, 12$), the data cell mark corresponds to the name of its corresponding vertex i, and its data fiber corresponds to all possible colorings of vertex i. The data fiber types for each data x_i correspond exactly to the ith column in Table 9.1. Specifically, the data fibers are as follows: $\Im_1 = \{r_1\}, \Im_2 = \{y_2\}, \Im_3 = \{b_3\}, \Im_4 = \{g_4\}, \Im_5 = \{r_5\}, \Im_6 = \{r_6, b_6\}, \Im_7 = \{b_7, g_7\}, \Im_8 = \{y_8, b_8, g_8\}, \Im_9 = \{y_9, b_9, g_9\}, \Im_{10} = \{y_{10}, g_{10}\}, \Im_{11} = \{r_{11}, y_{11}\}, \Im_{12} = \{y_{12}, b_{12}\}$.

(2) Probe Library Construction Probes are designed based on each edge in the graph. For convenience, we use x_i to represent the vertices $i \in \{1, 2, \ldots, 12\}$ from Fig. 9.9. As an example, consider the edge x_4x_{12}. Since vertex x_{12} has two possible colorings, the corresponding probe operator for edge x_4x_{12} has two possible probes: $\overline{g_4y_{12}}$ and $\overline{g_4b_{12}}$. In this way, probes can be designed for all the other edges. These probes are organized into probe sub-libraries, as shown in Table 9.2. Each sub-library corresponds to a specific edge, denoted as Y_{it}, where $\{i, t\} \subset \{1, 2, \ldots, 12\}$, and x_ix_t represents an edge in the graph shown in Fig. 9.9. For instance, the part corresponding to the edge x_1x_2 is $\overline{r_1y_2}$, indicating that the probe operator for this edge contains only $\overline{r_1y_2}$. Since the graph in Fig. 9.9 contains 30 edges, there are a total of 30 sub-probe libraries comprising 73 probes.

(3) Probe Operation Steps

Step 1: Create 12 types of nanoparticles (2.5 nm in diameter) as the 12 types of data cells. Encode the DNA sequences corresponding to 21 types of data

Table 9.2 The probe operator corresponding to each edge in the graph G, there are 30 sub-probe libraries, 73 probes

x_1x_2	x_1x_3	x_1x_7	x_1x_8	x_1x_9	x_1x_{10}	x_2x_3	x_2x_4	x_2x_6	x_2x_7	x_3x_4
$\overline{r_1y_2}$			$\overline{r_1y_8}$	$\overline{r_1y_9}$	$\overline{r_1y_{10}}$			$\overline{y_2r_6}$		
	$\overline{r_1b_3}$	$\overline{r_1b_7}$	$\overline{r_1b_8}$	$\overline{r_1b_9}$		$\overline{y_2b_3}$		$\overline{y_2b_6}$	$\overline{y_2b_7}$	
		$\overline{r_1g_7}$	$\overline{r_1g_8}$	$\overline{r_1g_9}$	$\overline{r_1g_{10}}$		$\overline{y_2g_4}$		$\overline{y_2g_7}$	$\overline{b_3g_4}$
x_3x_{10}	x_3x_{11}	x_4x_6	x_4x_{11}	x_4x_{12}	x_5x_7	x_5x_8	x_5x_9	x_5x_{10}	x_5x_{12}	x_6x_7
$\overline{b_3y_{10}}$	$\overline{b_3r_{11}}$		$\overline{g_4r_{11}}$	$\overline{g_4b_{12}}$	$\overline{r_5b_7}$	$\overline{r_5y_8}$	$\overline{r_5y_9}$	$\overline{r_5y_{10}}$	$\overline{r_5y_{12}}$	$\overline{r_6b_7}$
$\overline{b_3g_{10}}$	$\overline{b_3y_{11}}$	$\overline{g_4r_6}$	$\overline{g_4y_{11}}$	$\overline{g_4y_{12}}$		$\overline{r_5b_8}$	$\overline{r_5b_9}$		$\overline{r_5b_{12}}$	$\overline{r_6g_7}$
		$\overline{g_4b_6}$			$\overline{r_5g_7}$	$\overline{r_5g_8}$	$\overline{r_5g_9}$	$\overline{r_5g_{10}}$		$\overline{b_6g_7}$
x_6x_{12}	x_7x_8		x_7x_{12}	x_8x_9		x_9x_{10}		$x_{10}x_{11}$	$x_{10}x_{12}$	$x_{11}x_{12}$
$\overline{r_6y_{12}}$	$\overline{b_7y_8}$	$\overline{g_7b_8}$	$\overline{b_7y_{12}}$	$\overline{y_8b_9}$	$\overline{b_8g_9}$	$\overline{y_9g_{10}}$	$\overline{g_9y_{10}}$	$\overline{y_{10}r_{11}}$	$\overline{y_{10}b_{12}}$	$\overline{r_{11}y_{12}}$
$\overline{r_6b_{12}}$	$\overline{b_7g_8}$		$\overline{g_7y_{12}}$	$\overline{y_8g_9}$	$\overline{g_8y_9}$	$\overline{b_9y_{10}}$		$\overline{g_{10}r_{11}}$	$\overline{g_{10}y_{12}}$	$\overline{r_{11}b_{12}}$
$\overline{b_6y_{12}}$	$\overline{g_7y_8}$		$\overline{g_7b_{12}}$	$\overline{b_8y_9}$	$\overline{g_8b_9}$	$\overline{b_9g_{10}}$		$\overline{g_{10}y_{11}}$	$\overline{g_{10}b_{12}}$	$\overline{y_{11}b_{12}}$

fibers and then synthesize the corresponding DNA strands. Next, embed the DNA strands (data fibers) into the corresponding nanoparticles (data cells).

Step 2: Based on Step 1, construct a probe library containing 73 types of probe operators.

Step 3: Select an appropriate quantity of 12 data types from the database and 73 types of probe operators from the probe library. Add these to the computing platform. Various types of data aggregates are formed through specific hybridization reactions between DNA molecules and under the action of the computing platform. It should be noted that the current computing platform cannot yet satisfy the three basic functions mentioned earlier. The design of a computing platform capable of supporting these functions requires further research. However, for the nano-DNA probe machine model, even without these three basic functions, a sufficient quantity of data and probes allows for the formation of all true solution data aggregates.

Step 4: Using detection technology, detect all 12-order data aggregates that are isomorphic to the graph shown in Fig. 9.9. These data aggregates represent the true solutions to the problem, i.e., the 4-coloring of the graph. It is important to emphasize that current electron microscopes can only detect data aggregates of smaller orders and have difficulty detecting larger-order data aggregates. This limitation is a key reason why this model is not yet practical.

Theoretically, a single probe operation can identify all 14 true solutions of the graph in Fig. 9.9 (i.e., all 4-colorings). Figure 9.10 illustrates these colorings.

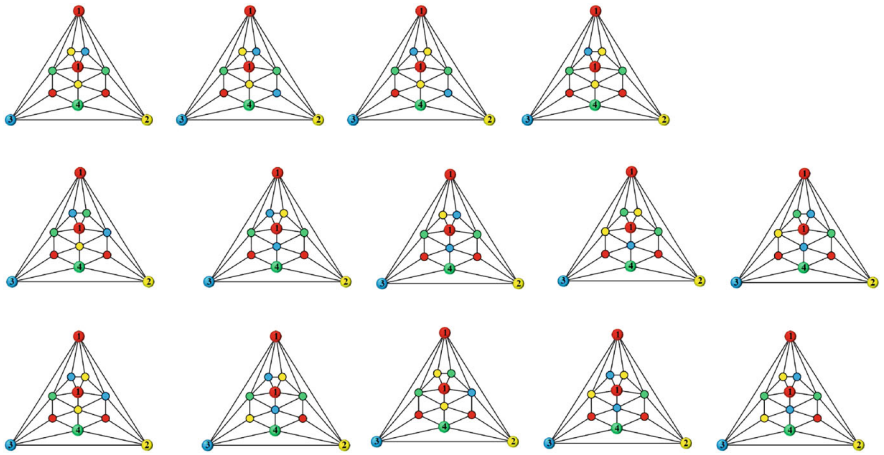


Fig. 9.10 All 14 types of 4-coloring of Fig. 9.9 [1]

9.5 Transmissive Probe Machine and Biological Neural Network

The probe machine model is inherently present in nature, with a typical example being the biological neural system, which can be considered a transmissive probe machine model. In the biological neural system, various types of neurons exist. Some neurons persist throughout the organism's life, such as those in the cerebral cortex, while certain brain regions contain stem cells capable of generating new neurons to replace those that die or undergo apoptosis. Conversely, there are neurons that, once lost due to death or apoptosis, are not replaced by new neurons. This dynamic behavior demonstrates that the neural network is constantly evolving over time. Information transmission between neurons occurs through synapses, which are classified into two main types: electrical and chemical. Information transmission in chemical synapses is unidirectional and relies on chemical substances (neurotransmitters) as the transmission medium. On the other hand, electrical synapses are typically bidirectional, with information transmission occurring through electrical currents (electrical signals). Regardless of the type, the transfer of information between neurons depends on action potentials, which we can consider as the fundamental probe operations in this model. Neurons themselves can be viewed as data, while neurotransmitters serve as the information transmitted within the probe machine.

From a mathematical model perspective, the probes of the probe machine can be categorized into two types: *connection-type probe operators* and *transmission-type probe operators*. The concept of the transmission operator arises from the observation of information transmission between neurons in natural biological systems facilitated by action potentials that travel through synapses. In this context,

action potentials can be regarded as transmission probes. Although the transmission-type probe machine mimics the information transmission in biological systems, it possesses two key differences that distinguish it from biological systems. First, in a biological system, the positions of any two neurons are fixed, whereas in a transmission-type probe machine, the positions of data fibers are variable. Information transmission only occurs when the machine locates two data fibers on the computing platform. Second, in biological systems, the connections between neurons are sparse—there are approximately 10^{12} neurons in the human brain but only 10^{15}–10^{16} synapses—while in the probe machine, the connections between data are extremely dense, and any two data could potentially be connected.

9.6 Probe Machine Function Analysis

According to the definition of a Turing machine, the set consisting of all Turing machines is countable, while the set of languages, denoted by Σ, is uncountable. This implies that the size of the set of languages is strictly greater than that of the set of Turing machines. Consequently, there exist languages that cannot be recognized by Turing machines, which indicates the existence of unsolvable problems, such as the halting problem [25]. Moreover, there is a one-to-one correspondence between the set of all languages and the set of real numbers. Similarly, a one-to-one correspondence exists between the set of all Turing machines and the set of rational numbers. From this comparison, it becomes evident that problems that Turing machines cannot recognize are abundant.

9.6.1 Turing Machines Are a Special Case of Probe Machines

In each operation of a Turing machine, the read-write head moves one cell to the left or right, reads the symbol in the current cell, deletes the symbol, or writes a new symbol into the cell. The execution of this process is governed by the state transition function $\delta(Q, \Gamma)$, where Q is the set of states and Γ is the set of available symbols on the storage tape. When defined, $\delta(Q, \Gamma)$ is represented as a triplet (p, Y, D), where $p \in Q$ is the next state, $Y \in \Gamma$ is the symbol written on the cell pointed to by the current read-write head, and D indicates the direction in which the read-write head moves, either left or right. The following argument demonstrates that the probe machine can simulate the state transition function $\delta(Q, \Gamma)$ of a Turing machine, thereby proving that a Turing machine is a special case of a probe machine.

In the probe machine, the data consists of two components: data cells and data fibers. Each data cell is an information source with multiple data fibers, as illustrated in Fig. 9.3a. For simplicity, we consider the case where each data cell contains only two types of data fibers, as shown in Fig. 9.11a. In this case, we select a representative for each type of data fiber, as depicted in Fig. 9.11b. In this

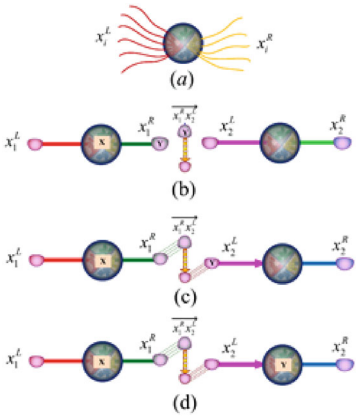


Fig. 9.11 Illustrative example of a probe machine simulating a Turing machine [1]

example, the data cell x_1 contains information X, and its two types of data fibers are labeled x_1^L and x_1^R. Similarly, the data cell x_2 has two types of data fibers, x_2^L and x_2^R. The probe operator between the data fibers x_1^R and x_2^L is denoted as $\overline{x_1^R x_2^L}$. Corresponding to the state transition function $\delta(Q, \Gamma)$ of a Turing machine, we assume, without loss of generality, that the read-write head of the Turing machine is instructed to move left. At this point, the data fibers x_1^R and x_2^L are connected by the probe operator $\overline{x_1^R x_2^L}$, as shown in Fig. 9.11c. If x_1^R carries information, this information will be transmitted to the data cell of x_2, as illustrated in Fig. 9.11d.

9.6.2 Can Turing Machines Simulate Probe Machines?

The previous section demonstrated that probe machines can simulate Turing machines. However, the question arises: Can Turing machines simulate probe machines in return? To address this, we begin by revisiting the definition of probe machines and examining their fundamental functions. Let $X' \subseteq X$ and $Y' \subseteq \tau(X')$, where the result of applying the probe operation $\tau(X', Y')$ to the pair (X', Y') is denoted as Θ. The set Θ may contain a single true solution or multiple true solutions, which are topologically related through isomorphism to the graph $G^{(X',Y')}$, but the weights assigned to the corresponding vertices may differ. Here, the weight of a vertex is defined as the data fiber associated with that vertex. During a probe operation, the number of basic probe operations corresponds exactly to the sum of the number of edges in the graphs representing all the solutions. This parallelism in the underlying computation of the probe machine is what allows it to solve many **NP**-complete problems with minimal probe operations, such as the Hamiltonian problem, graph coloring, SAT problem, TSP, maximum clique, minimum independent set, and vertex cover problems, among others. In 1971, Cook et al. [26] demonstrated that all **NP**-complete problems, in the context of a Turing

machine, are polynomially equivalent. This implies that, in the context of a probe machine, these problems are no longer **NP**-complete. Each operation of a Turing machine involves moving the read/write head one cell to the left or right, erasing the current symbol, and writing a new symbol. From a topological perspective, the execution of a Turing machine corresponds to a path of length 1, which relates to only a single edge of a given graph (solution). Thus, exploring how a Turing machine can simulate a probe machine is a task of great significance.

9.6.3 Advantages of the Probe Machine

Based on the previous discussions, the probe machine offers several notable advantages. One of the primary strengths of the probe machine is its ability to exponentially increase its information processing capability as the size of the database grows. Specifically, the information processing capacity of a probe machine during a single probe operation is 2^q, where q represents the total number of edges across all graphs isomorphic to $G^{(X',Y')}$. This implies that for a data set of size $n = 50$, and assuming each data element can be probed, the processing capability of the probe machine reaches $2^{25\times 49} = 2^{1225}$. Such a vast computational capacity is sufficient to break public key cryptosystems potentially.

9.7 Conclusion

The Turing machine processes adjacent data sequentially, and its data placement mode is linear, which inherently restricts it to a serial computing model. In contrast, the data placement mode of the probe machine is spatially unrestricted, allowing any two data elements to be adjacent. Any data pair can directly process information, providing the probe machine with inherent parallelism. As a result, the probe machine typically requires only one or a few probe operations to solve problems.

The probe machine consists of nine components, the most crucial being the database and the probe library. The basic probe operations are categorized into connection-type and transmission-type, depending on the kind of fiber attached to the data in the database. A connection-type probe operation processes information between two data fibers without considering direction, while a transmission-type probe operation is direction-dependent. We hypothesize that these are the only two basic probe operations. For example, in solving the Hamiltonian problem and the graph vertex coloring problem, the probe operation is of the connection type. Meanwhile, information processing between biological neurons falls under the transmission-type probe operation. This raises the question: Are there other basic probe operations beyond these two types? Is there a hybrid probe machine that incorporates both connection and transmission probes? These intriguing questions warrant further thought and exploration.

As a mathematical model, the probe machine theoretically offers significant advantages over the Turing machine in solving many complex problems. Consequently, computers based on probe machines could play a key role in developing human society and even advancing civilization. A practical concern, however, is the choice of material for building computers based on probe machines. While a nano-DNA probe machine model has been proposed, it is still far from practical, given the current limitations of detection technology. In contrast, the transmission-type probe machine seems more promising for realization [19, 20]. We believe that the transmission-type probe machine has the potential to surpass human cognitive capabilities in certain areas.

References

1. Xu, J.: Probe machine. IEEE Transactions on Neural Networks and Learning Systems **27**(7), 1405–1416 (2016)
2. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. J. Math. **58**(345–363), 5 (1936)
3. Turing, A.M.: Computability and λ-definability. The Journal of Symbolic Logic **2**(4), 153–163 (1937)
4. Moore, G.E.: Progress in digital integrated electronics. In: International Electron Devices Meeting, Washington, D.C., IEEE, vol. 21, pp. 11–13 (1975)
5. Feitelson, D.G.: Optical Computing: A Survey for Computer Scientists. MIT Press, Cambridge, MA, USA (1988)
6. McAulay, A.D.: Optical Computer Architectures: The Application of Optical Concepts to Next Generation Computers. John Wiley & Sons, Inc., USA (1991)
7. Witlicki, E.H., Johnsen, C., Hansen, S.W., et al.: Molecular logic gates using surface-enhanced Raman-scattered light. Journal of the American Chemical Society **133**(19), 7288–7291 (2011)
8. DiVincenzo, D.P.: Quantum computation. Science **270**(5234), 255–261 (1995)
9. Li, J.S., Li, Z.B., Yao, D.X.: Quantum computation with two-dimensional graphene quantum dots. Chinese Physics B **21**(1), 017302 (2012)
10. Li, H.O., Yao, B., Tu, T., et al.: Quantum computation on gate-defined semiconductor quantum dots. Chinese Science Bulletin **57**, 1919–1924 (2012)
11. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(5187), 1021–1024 (1994)
12. Roweis, S.T., Winfree, E., Burgoyne, R., et al.: A sticker based model for DNA computation. In: DNA Based Computers, Proceedings of a DIMACS Workshop, Princeton, New Jersey, USA, pp. 1–29 (1996)
13. Winfree, E., Liu, F., Wenzler, L.A., et al.: Design and self-assembly of two-dimensional DNA crystals. Nature **394**(6693), 539–544 (1998)
14. Mao, C., LaBean, T.H., Reif, J.H., et al.: Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature **407**(6803), 493–496 (2000)
15. Douglas, S.M., Marblestone, A.H., Teerapittayanon, S., et al.: Rapid prototyping of 3D DNA-origami shapes with caDNAno. Nucleic Acids Research **37**(15), 5001–5006 (2009)
16. Xu, J., Qiang, X., Yang, Y., et al.: An unenumerative DNA computing model for vertex coloring problem. IEEE Transactions on Nanobioscience **10**(2), 94–98 (2011)

17. Xu, J., Qiang, X., Zhang, K., et al.: A DNA computing model for the graph vertex coloring problem based on a probe graph. Engineering **4**(1), 61–77 (2018)
18. Hu, M., Li, H., Chen, Y., et al.: Memristor crossbar-based neuromorphic computing system: A case study. IEEE Transactions on Neural Networks and Learning Systems **25**(10), 1864–1878 (2014)
19. Duan, S., Hu, X., Dong, Z., et al.: Memristor-based cellular nonlinear/neural network: design, analysis, and applications. IEEE Transactions on Neural Networks and Learning Systems **26**(6), 1202–1213 (2014)
20. Gong, M., Liu, J., Li, H., et al.: A multiobjective sparse feature learning model for deep neural networks. IEEE Transactions on Neural Networks and Learning Systems **26**(12), 3263–3277 (2015)
21. Xia, Y., Wang, J.: A bi-projection neural network for solving constrained quadratic optimization problems. IEEE Transactions on Neural Networks and Learning Systems **27**(2), 214–224 (2015)
22. Hamilton, W.R.: Letter to John Graves on the Icosian, 17 Oct., 1856. The Mathematical Papers of Sir William Rowan Hamilton **3**, 612–625 (1931)
23. Xu, J., Li, F.: Principles, progress, and challenges of DNA computers (V): DNA molecule immobilization technology. Chinese Journal of Computers **32**(12), 2283–2299 (2009)
24. Cohen, R., Schmitt, B.M., Atlas, D.: Reconstitution of depolarization and Ca2+-evoked secretion in Xenopus oocytes monitored by membrane capacitance. Exocytosis and Endocytosis, pp. 269–282 (2008)
25. Sipser, M.: Introduction to the Theory of Computation. ACM Sigact News **27**(1), 27–29 (1996)
26. Cook, S.A.: The complexity of theorem-proving procedures. In: Proceedings of the Third Annual ACM Symposium on Theory of Computing, pp. 151–158. ACM, New York, NY, USA (1971)

Chapter 10
DNA Algorithmic Self-Assembly

Since Seeman proposed using DNA as a nanomaterial, DNA Tile and DNA origami technologies have been successively proposed, and DNA has realized the programmable assembly from two-dimensional patterns to three-dimensional spatial structures. The nanostructures constructed by DNA have stable properties, regular geometric appearance, and spatial addressability, and are widely used in many fields. DNA algorithmic self-assembly can be realized by controlling the self-assembly process in ways such as sequence and connection rules. With the programmability of DNA sequences, the design of DNA nanostructures to realize DNA algorithmic self-assembly has made good progress. This chapter mainly includes DNA Tile computation, Turing equivalent Tile computation, programmable Tile structure, single-strand Tile (SST) computation, universal DNA Tile computation, DNA origami computation, etc.

10.1 DNA Tile Computation

DNA Tile is an important concept in the field of DNA nanotechnology. It is a self-assembled nanostructure formed by precisely designed nucleic acid sequences. Each DNA Tile is composed of a series of short DNA strands, which are combined with each other through the principle of base pairing to form a stable two-dimensional or three-dimensional structure. The design of DNA Tile is usually based on the "molecular template" principle, where one DNA molecule coding region (called the template) determines the arrangement and combination of another molecule coding region or a set of molecule coding regions. The specific DNA sequence coding design is based on the principle of specific base complementary pairing, and the single-stranded sticky ends exposed by the DNA Tile structure guide the Tile to automatically assemble into a predetermined geometric shape. These Tiles can further assemble into larger structures, such as lattices or other complex nanoscale patterns and devices.

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_10

The tiling of DNA Tiles is to use the basic DNA molecular structure as a tile or ceramic tile (Tile) to cover the ground like laying tiles, or to piece together a specific shape like building blocks. This embodies the bottom-up self-organization idea, that is, to use DNA Tile as the basic unit, and gradually expand and piece together into a specific pattern according to certain rules. By encoding the given DNA molecule sticky end splicing logic and restriction conditions to the DAN sequence, DNA can Tile forms regular patterns according to the set method, this programmable self-assembly technology reflects the computability of DNA Tile. DNA Tile is one or several basic structures, this programmable bottom-up self-assembly method is also the most important idea of DNA Tile self-assembly.

The programmable self-assembly of DNA Tile provides the physical basis for the implementation of DNA Tile computation. DNA Tile assembly is the process of building physical structures, and computation is the logic and algorithm operations implemented on these structures. Assembly is the basis of computation, and computation gives assembly higher functions and purpose.

10.1.1 DNA Tile Types

The design of DNA Tile was inspired by the DNA Holliday junction structure in nature. Researchers designed a very simple but effective structure called the four-arm junction. As shown in Fig. 10.1, it can connect in four directions: up, down, left, and right. Each chain does not fully combine with another chain, and the exposed ends can cascade. In this way, the four-arm structures can connect with each other to form a large network structure. On this basis, improvements can be made to the four-arm junction to form a more stable structure, and the new structure can be applied to the design of finite arrays.

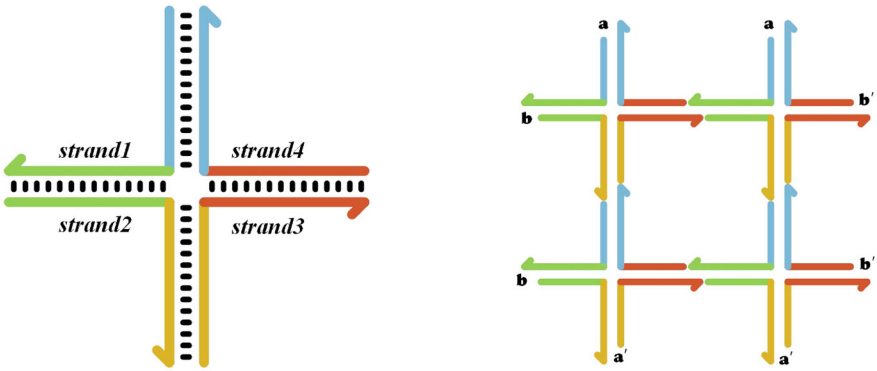


Fig. 10.1 (left) Four-arm junction Tile. (right) Four-arm junction connects in four directions: up, down, left, and right

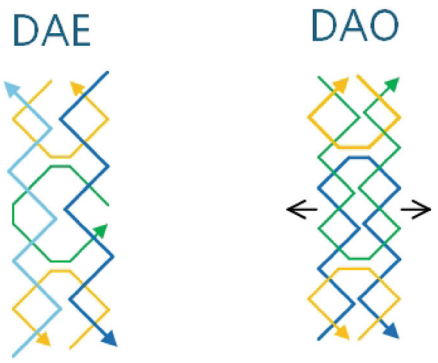


Fig. 10.2 Stable structures DAE/DAO in DX

Although the four-arm junction Tile appears to be 4-fold symmetric, in fact, with the rotation of the structure, the structure is only 2-fold symmetric. Moreover, due to the relaxation of the structure, the four-arm junction Tile is not very stable. Based on this, Seeman and others developed the DX Tile [1], which contains two crosses, is made up of two strands of DNA double helix side by side, and each structure contains 4–5 single strands. Fu and others designed several different DX Tile structures [2]. However, after experimental verification, only two are relatively stable, namely DAO and DAE, as shown in Fig. 10.2.

The naming method of these two structures represents their structural characteristics. D represents that it is a double cross (Double Helix), meaning that these two structures are both double helix structures; A represents that the cross point is antiparallel; O represents that the number of helical pitches spanned by the two cross points of the adjacent double helix is odd (Odd), and E is even (Even). In fact, when designing the DNA Tile structure, the relationship between the pitches should be noted. It is required that each chain does not produce torque when spanning the pitch, so the best effect is for the single chain to span at the integer of the pitch. In fact, because one pitch is about 10.5–10.6 bases, there will be a certain error between the design scheme based on DNA Tile self-assembly and the actual situation. That is, each pitch accumulates about 5 degrees. When designing the DNA Tile structure, the deviation of 5 degrees can almost be ignored. And in the self-assembly process, the twist can be cancelled by the connection form of positive and negative.

The success of DX Tile laid a solid foundation for its large-scale assembly. The literature [3] changes the internal and lateral spacing of the Tile, allowing the Tile to misalign in the same direction. Thus, the figure is changed from one-dimensional to two-dimensional complex structure (Fig. 10.3). This improvement can also realize the self-assembly of complex multi-Tile systems, as well as stripe modification and streptavidin protein chimera complex labeling. It laid a solid foundation for later self-assembly based on DNA origami.

The DX structure is relatively small, so the structure is stable, and it has high robustness when connected into a large-scale structure. However, because DX and TX only extend sticky ends in two directions for connection, their connection directions are restricted and cannot be expanded in other directions. In 2003, Yan

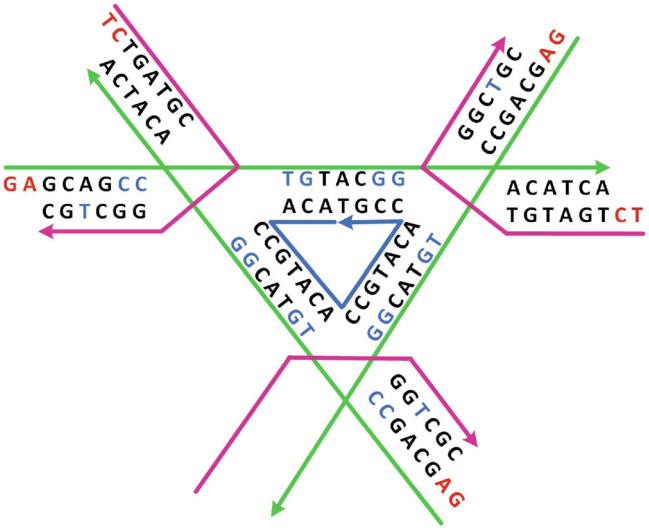


Fig. 10.3 3D DNA tile

Hao and others synthesized the advantages of the above DX/TX structure and invented the 4 × 4 Tile, also known as the cross Tile. It has 4 arms that can connect in four directions: up, down, left, and right. Each arm has a dedicated chain for stabilizing the structure, and each arm forms a cross with the common center chain in the center. This structure can form a stable cross, which can make the entire structure very stably connected in four directions to form a stable square network structure [4]. This structure can not only connect in four vertical directions, but also form connections in any other plane direction by adjusting the angle between the arms, realizing some special applications. Such as molecular printing, finite grid, nano-marking, etc.

Seeman et al. extended the above Tile, they designed each arm in the cross Tile to be completely symmetrical arms, and obtained 3-arm/8-arm/12-arm structures [5, 6]. They call it the N Tile series, where N represents the number of Tiles. N Tile is a general-purpose Tile system, theoretically each Tile only needs three DNA single strands to form. Tile system can assemble nano-scale regular grids and symmetrical three-dimensional structures, these structures are precise and regular, reflecting the powerful self-assembly ability of N Tile.

In fact, the Tile system can continue to be subdivided downwards, and Sub-Tile can be formed through the assembly between arms. In 2014, the literature [7] assembled perfect nanotubes and nanobelts through the Sub-Tile strategy, reflecting the bottom-up programming ability of the Tile self-assembly system.

The design of the molecular Tile system requires not only mathematical theory proof but also experimental verification. At present, there are dozens of small molecular Tiles designed theoretically, but the commonly used ones in actual experiments are the experimentally verified mature schemes mentioned above.

10.1.2 DNA Tile Calculation Example

Using the above Tile system, researchers can form one-dimensional, two-dimensional, or three-dimensional nanostructures. Through one or several Tiles freely combining in solution, periodic network structures are formed. These Tiles are not subject to connection restrictions, and they form infinite patterns through periodic connections with themselves or other Tiles.

In 2005, literature [8] used the improved cross Tile to form a 3 × 3 grid and extended probes on the grid to achieve specific hybridization. At the same time, the LaBean group designed a 4 × 4 grid, specifically biotin and streptavidin labeling on the grid, achieving 5 nm level DNA pattern labeling [9].

Depending on the coding of each Tile, the final network structure can be solid or hollow. It can even grow and arrange according to specific rules, achieving stop or judgment and other rule-based connections. For example, by adding different stop growth rules, 2 × 2, 3 × 3, 4 × 4 different sizes of network structures can be generated. If the stop growth rule is defined in the center of the network, different sizes of frames can be generated.

Labean and others used the improved cross Tile to design a fully addressable and precisely programmable array. Through step-by-step hierarchical assembly technology, a fully addressable finite-sized array [9] was manufactured. Each arm of this DNA Tile contains a DNA Holliday structure, and each cross structure can ensure the stability of the arm. Observed by atomic force microscopy, the size of the nanostructure is completely consistent with the design, and the length of a single Tile is about 10nm. The DNA nanoarray they designed can be used as a development template for organizing heterogeneous materials.

Through the assembly of cross-shaped Tiles, Labean and others have verified an easy-to-develop universal assembly system. Through

Nanotechnology and molecular chemistry control the structure at the nanoscale and use self-assembly to build fully addressable, finite-sized arrays, demonstrating a variety of self-assembly program modes. In addition, they also decorated the assembly structure with proteins shaped like the letters “D”, “N”, and “A”, demonstrating the fully addressable nature of the structure.

The displacement connection of DX Tile to realize the cumulative XOR logic gate is a classic example. Rothemund and others achieved a 9-bit addition, and the structure displayed under the atomic force microscope is a complete Sierpinski triangle pattern [10]. This scheme is the first to use small molecule DNA Tiles for mathematical calculation design and experiments. The experimenters not only proved the computational efficiency of the system from mathematical principles but also discussed the possible causes of errors in the experiment. As shown in Fig. 10.4.

Rothemund and others used the tile assembly model to realize the classic Sierpinski triangle structure. They used the two-dimensional self-assembly technology of DNA tiles, based on the update rules of the automaton to calculate binary. They formed a Sierpinski triangle, a Sierpinski triangle is a figure formed by the accumulation of triangles, and as the structure grows, the XOR logic makes

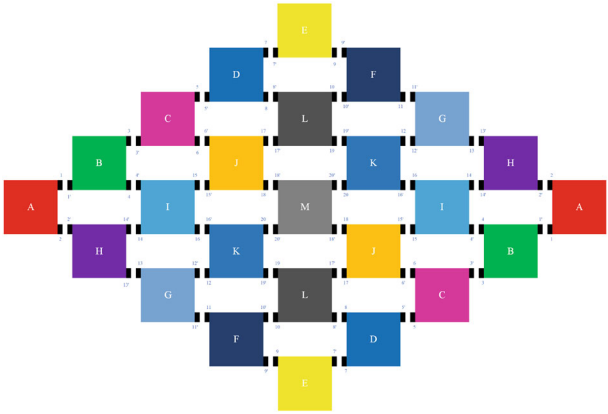


Fig. 10.4 Tile implement the 5 × 5 matrix

the structure form a fractal pattern. To realize the Sierpinski triangle, the abstract triangle structure is concretized as a DNA tile of a double-cross pattern. As an input for computation, long single-strand DNA molecules are used to grow tiles into cumulative structures. The assembly error rate of their formed Sierpinski triangle is between 1% and 10%. Although not perfect, the generation of the Sierpinski triangle structure demonstrates all the necessary mechanisms for any DNA molecular automaton to implement computation, that is, a simple paste model. This shows that the DNA paste model can be treated as a Turing universal biomolecular system, capable of partially computing or constructing task algorithms.

They use the protrusions and depressions of the tile structure to distinguish between 0 and 1. According to the values of the four corners of the tile structure, only four different computing modules are needed to assemble the Sierpinski triangle.

The splicing of DNA blocks actually completes the calculation of the cumulative XOR logic gate. The self-assembly of the DNA Sierpinski triangle demonstrates all four features required for Turing equivalent computation: the formation of structural crystals, the formation of extended crystals, programmable interaction between DNA tiles, and the templating of input and output information.

In the experiment, Rothemund and others used two different DNA tile structures, DAE and DAO. In the experimental design of DAE, the top center includes four DAE-E molecules, namely VE-00, UE-11, RE-01, and SE-10. Each symbol '0' or '1' is represented by making the complementary shape large or small. The pink legend shows the mapping from shape to paste sequence. The molecular diagram of VE-00 shows how each DAE-E tile is composed of five DNA strands. Six DAO-E computing units are used in the design of DAO: S-00, R-00, S-11, R-11, S-01, and R-01. The symmetry of DAO-E molecules allows each molecule to be used for a computing module, and the two modules S-00 and R-00 are used to restrict the input structure.

The result of the atomic force microscope found that the connection efficiency of DAE is not as high as that of DAO, but in terms of structural mismatch, DAE has more reliable computing power. In fact, the study of algorithmic self-assembly can be further understood as simple biological self-assembly. Algorithmic crystals composed of simple DNA tiles serve as templates for simplicity and multifunctionality. However, biological self-assembly is more complex and precise, including conformational changes, dissipation mechanisms, ATP hydrolysis, genetic interactions, etc.

The interaction between DNA tiles is determined by the selective association of sticky end sequences. Well-designed initial input layer tiles grow in a controlled manner triggered by input information. This tiling method can be used to implement other cellular automaton rules. DNA tile self-assembly is programmable and Turing equivalent. In addition, for manufacturing purposes, the direction and degree of growth can be controlled by self-assembled calculations, allowing efficient creation of arbitrary shapes. The main obstacle currently limiting self-assembly calculations is errors during the growth process, which is limited by current molecular biotechnology. There are several types of errors in the experiment, including lattice dislocations, structural errors of mismatched tile connections, and errors during the growth process. Developing an accurate quantitative model is very valuable for algorithmic self-assembly.

10.2 Turing Equivalent Tile Calculation

The mathematical definition of DNA Tile calculation can be defined from the perspective of computer algorithms and mathematical logic. DNA Tile calculation involves transforming computable problems into a series of programmable molecular self-assembly processes. The Tile calculation system can theoretically be designed as a Turing-complete system, capable of simulating any computable algorithm. Specifically, by properly designing DNA Tiles and their interaction rules, a computable system that performs any Turing equivalent complex algorithm can be constructed.

DNA Tile calculations are often described through programmable molecular self-assembly models, which define how DNA Tiles self-assemble into larger structures based on their edge complementarity. Each type of Tile can be regarded as a "state", and their programmable connections simulate the state transitions of the Turing calculation system.

10.2.1 Mathematical Model of DNA Tile Calculation

The mathematical model of DNA Tile calculation originates from the floor tiling problem in the 1960s. In 1961, the Chinese-American mathematician Wang Hao

proposed a mathematical problem based on floor tile assembly—Wang Tile [11] in the Bell System Technical Journal, that is, whether there is a set of tiles that can non-periodically tile the entire plane.

The set of tiles stipulated by Wang Tile is a polygonal figure with a certain rule shape and different colors. The basic rule is that edges of the same color can be assembled and connected together, and they must be shape-matched, that is, the corresponding edges can be nested. According to this rule, a set of Tiles are assembled one by one to form a plane or a network structure. This calculation process is Tile self-assembly, and Tile calculation is also Turing equivalent.

Intuitively, each DNA Tile has a connection direction, on its east, south, west, and north sides. Connections occur when the matching strength on adjacent edges is greater than the stability of the current single block. This connection method belongs to entropy-driven edge connection and does not require coding of sticky ends. The connection method based on base complementary pairing involves algorithm design. Winfree gave a complete definition of the self-assembly connection model [12]. And a series of subsequent experiments validated this model [13, 14]. DNA Tiles can contact a large number of other Tiles under the drive of molecular thermal motion, and the sticky ends of Tile DNA chains connect under the constraint of the base complementary pairing rule. It should be noted that the stability of this connection is determined by the entropy stability of the connecting molecules and the temperature in the external environment. When the stability of the molecular connection is greater than the environmental temperature, the connection is stable. The definition and operation of the DNA Tile system are as follows.

The four connection domains of DNA Tile can be defined as a finite set Σ, and the connection rules of the connection domain are defined as μ. When a Tile is not connected by other Tiles, the Tile can be defined as "empty". $\boldsymbol{T}$ is a set of specific Tile sets, and τ is the energy required for connection (generally molecular thermal drive).

In the connection set Σ, the empty set is also included, that is $null \in \Sigma$. For a Tile, the operation of four-direction connection is denoted as δ, and the Tile is denoted as $\langle \delta\ N\ \delta\ W\ \delta\ E\ \delta\ S \rangle \in \Sigma$. The direction function set here is $\{N, W, E, S\}$ to implement position mapping. For a certain position $(x, y) \in z$, the direction set D can be represented as:

$$\begin{aligned} N(x, y) &= (x, y + 1); \quad W(x, y) = (x - 1, y); \\ E(x, y) &= (x + 1, y); \quad S(x, y) = (x, y - 1) \end{aligned} \tag{10.1}$$

The definition of two adjacent Tiles (x, y) and (m, n) is denoted as:

$$(x, y), (m, n) \in Z; \quad \text{if } d \in D,\ d(x, y) = (m, n) \tag{10.2}$$

The connection strength g is represented as:

$$\Sigma \times \Sigma \to R \tag{10.3}$$

The connection strength g is variable, where any connection operation $\forall\delta \in \Sigma$. For an empty edge, its connection strength is 0, $g\left(null, \delta\right) = 0$. Suppose:

$$g\left(\delta, \delta'\right) = 0 \Leftrightarrow \delta \neq \delta' \tag{10.4}$$

Equation 10.4 explains the definition of two adjacent Tiles $\left(\delta, \delta'\right)$ that cannot be connected from the connection strength. Therefore, defining $g = 1$ means $\forall\delta \neq null$, if $g\left(\delta, \delta\right) = 1$ and $\forall\delta \neq \delta'$ then $g\left(\delta, \delta'\right) = 0$.

If the set T is a group of Tiles containing empty edges. Then the operation formula for the position of T can be defined as A: $z \times z \rightarrow T$. For this position $(x, y) \in A$, if A $(x, y) \neq empty$. The connection of A is finite, only when there are only finite connected positions, that is $(x, y) \in A$.

Finally, a Tile system S is a triplet $< T, g, \tau >$. Here T is a Tile set containing the empty set, g is the connection strength, $\tau \in N$ represents the thermal energy required for connection.

If there is a Tile system S, $S =< T, g, \tau >$. S some Tiles A satisfy $T' \subseteq \Sigma^4$, under this circumstance, Tile ($t \in T$), t can be connected to A at position (x, y), and the condition for generating a new connection graph A' after connection is:

$$\begin{aligned}
&(x, y) \notin A, \quad \text{and} \\
&\Sigma_{d\in D}\, g\left(\mathrm{bd}_d\left(t\right), \mathrm{bd}_{d-1}\left(A\left(d\left(x, y\right)\right)\right)\right) \geqq \tau, \quad \text{and} \\
&\forall\left(u, v\right) \in Z^2, \left(u, v\right) \neq \left(x, y\right) \Rightarrow A'\left(u, v\right) = A\left(u, v\right), \quad \text{and} \\
&A'\left(x, y\right) = t
\end{aligned} \tag{10.5}$$

From Eq. 10.5, it can be seen that Tile t must meet several conditions to be able to connect to the graphic block. (1) t can only be connected at an empty position, that is, the position already occupied by the original graphic block cannot be connected again. (2) The connection energy of the connected edge τ must be greater than or equal to the energy required for base complementary pairing. (3) The stability entropy after connection needs to be greater than or equal to the temperature of the environment, otherwise the connection is unstable. For example, for any connection operation δ, if $g\left(\delta, \delta\right) = 1$, and $\tau = 2$, then the energy possessed by this Tile can be connected on two adjacent edges among $\{N, W, E, S\}$.

For a given Tile system $S =< T, g, \tau >$, and the set of Tile systems Γ. The seed graph S_0 (S_0 can be either the starting Tile or the starting graph): $Z^2 \rightarrow \Gamma$. If all the above conditions are met, then T is connected to S_0. Define $W_0 \subseteq Z^2$, W_0 is the set that can be connected to the upper direction. On this basis, another definition $w \subseteq W_0$, U_w is connected at w to the set of S_0. Let $\widehat{S_1}$ be all possible connection graphs, each possible connection edge probability is p, with $p \in S_0$, $S_1\left(p\right) = S_0\left(p\right)$ represents that the connection probability at these two places is the same.

For all $S_1 \in \widehat{S_1}$, it can be explained that the Tile system S is connected to S_0 in one step reaction, and the connection result is S_1. If there is a series of connection

results $\{A_0, A_1, \ldots, A_n\}$ are all connected graphs, $i \in \{1, 2, 3, \ldots, n\}$, and the Tile system S is connected from A_{i-1} to generate A_i in one step. In this way, it can be defined that S is connected from the seed graph A_0 to produce A_n in n-step reaction. If S can only generate A_n in n steps, then A_n is called the only final result. If from S_1 to S_n, the final result is A_0, and n is the minimum steps required to generate A_0. Then n is called the time required for assembly.

DNA Tile computation starts from a seed point S, and other Tiles gradually connect to S. If no other Tiles are connected to S, the computation ends. Under certain conditions, it is possible that multiple Tiles connect to a specific location, or a Tile can be connected by multiple other Tiles. The scheme that can be connected by multiple other Tiles can also form multiple different final connection results. In this case, we say that multiple connection results can be formed starting from S. Since a large number of Tiles are connecting at the same time, this process of large parallel computation will eventually produce the result with the minimum connection steps. This is also the reason why the connection result will eventually tend to thermodynamic stability.

10.2.2 Turing Equivalence of DNA Tile Computation

The computational power of DNA Tiles has been proven to be Turing equivalent [15], theoretically it can be used to construct an automatic molecular solution system for any computable function. Taking a one-dimensional cellular automaton as an example, it can be assembled by two Tile systems. 1-D cellular automata are Turing equivalent, and they have been proven to be able to perform Turing computation.

A 1-D cellular automaton is an infinitely long strip, and the state of the cells on the strip can be transformed according to the state of its neighboring cells. The 1-D tiling system represents a one-dimensional cellular automaton in a single row and can calculate the state of the automaton in the previous row in time. Through the transformation of each cell state, the 1-D cellular automaton can simulate a general one-dimensional cellular automaton.

Here we introduce a structure that converts a Turing machine into a Tile computing system. This system calculates the state description of the Turing machine in real time at each transformation step to simulate the Turing machine, and can well simulate the computing process of the 1-D Tile system.

A Turing machine is a simple computer. The exterior of the Turing machine contains an infinitely long data tape, a tape head that can read data, and the tape head can read a data cell on the data tape each time. The interior of the Turing machine is a state-limited data structure and state controller. The state controller can change the state of a data cell each time. The operation of the Turing machine follows some simple transformation rules, these simple rules are reflected in the state changes of the data cells (the data tape contains data cells, the state of the data cells). The tape head can convert a data cell from one state to another and write a symbol code on the data cell. After completing this transformation, the tape head can move to the

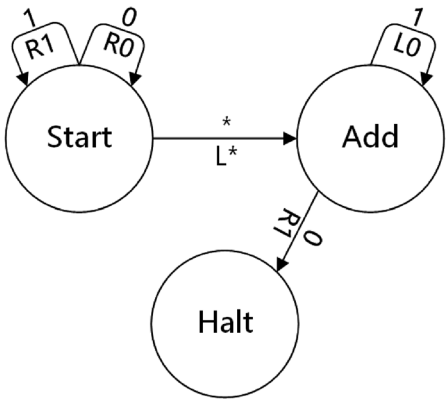


Fig. 10.5 Three-state Turing machine

adjacent data cell, to the left or to the right. The significance of the Turing machine is that it can complete all computing functions, and there is no more "powerful" computer that can surpass it. Here, powerful does not mean faster or more efficient, but refers to the computational theoretical structure.

Figure 10.5 shows the computing principle of a three-state Turing machine, which includes three operations (start, add, stop), and three states (represented as 0,1,*). When the tape head is placed on the data tape, it can write 0 or 1 on the data cell, or move left or right. "*" Indicates that the writing stops at this place, which is a sign of the stable operation state of the data cell at this place.

The L* in the figure indicates moving to the leftmost data cell and stopping moving to the left. Write 0 or 1 (Add) at this place. And continue to write 0 or 1 at other positions until it stops (Halt).

The DNA Tile computing system can simulate the computing process of the Turing machine. As shown in Fig. 10.6, the Tile system simulates the computing process of the Turing machine. This Tile system completes the calculation in the clockwise direction of the five-step process. Each row of Tiles represents a state of the Turing machine. The start of the yellow data is the bottom row of Tile tapes. It should be noted that the connection of Tiles needs to be connected layer by layer, and it cannot be broken in the middle, otherwise the final result cannot be obtained or the result obtained is wrong.

The calculation in Fig. 10.6 starts from a row of seed Tile strips (from left to right), and adds 1 to the value represented by the seed strip. Each step of the calculation process is reflected through the assembly of the next layer of Tiles. As shown in Fig. 10.6, starting from the upper left corner, in a clockwise direction, the Tile strips of the 2nd, 3rd, 6th, and 10th layers undergo state changes. The calculated value shown in the figure is 11 = 01011, and the final result is 12 = 01100. This Tile system has completed the calculation definition of the formula $f(a) = a + 1$. The red square block represents the position of the tape head on the data tape of the Turing machine. This position corresponds to the state of the Tile in the Tile system schematic, represented in red.

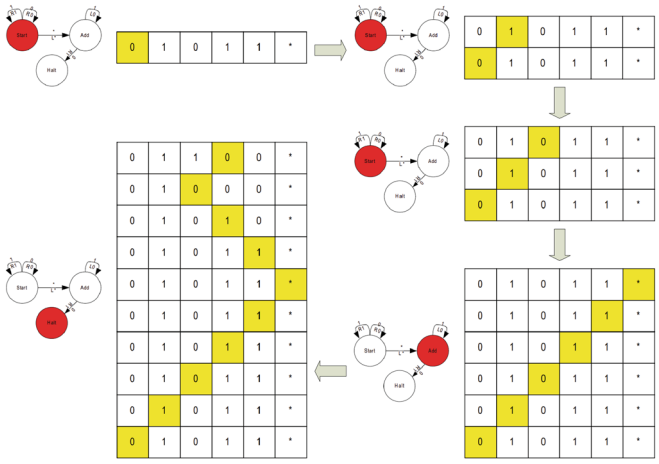


Fig. 10.6 DNA Tile system simulates the process of Turing calculation

Although this Tile system has only completed a simple addition, and used 10 layers of Tile strips, it reflects the process of Turing equivalence. In fact, the Tile system is not only Turing equivalent, but there is also a corresponding Tile system for each Turing system that can complete the corresponding operation. The number of Tiles required to convert a Turing system to a Tile system is

$$N = \Omega(|\varrho| \bullet |\Sigma|) \tag{10.6}$$

Here, ϱ represents the required finite control set, and Σ represents the letter state set. Other papers have also discussed the equivalence problem between the Tile system and the Turing machine. For example, Brun and others also proved that under a stable entropy, the Tile system is Turing equivalent [16].

With the improvement of the level of biomolecular biotechnology, people have a deeper understanding of the characteristics of DNA, and the control of DNA chains is more handy. The four bases of DNA can be freely swapped and encoded, and researchers can use restriction endonucleases, exonucleases, polymerases, ligases, etc. to operate on DNA, which can achieve more complex and rich calculations. And because the nanometer length of the DNA molecule chain can set a Tile as a short DNA chain or a double DNA chain, this helps to increase the order of magnitude of the number of Tiles. At the same time, more advanced instruments also provide great help to the calculation results.

Through the research of the above work, it can be concluded that the Tile system is Turing equivalent. The advantage of the Turing computer lies in the supporting electronic components, diodes, resistors, chips and other hardware to support its work. And with the development of the Turing machine, a series of software can also cooperate to write high-level languages, such as Java, C, C++, etc. The combination of these hardware and software is conducive to the construction of very complex

computing systems. Therefore, the research on the Tile system should not only stay at the simple addition and subtraction multiplication method of Turing equivalence, but should fully utilize the large-scale computing nature of the Tile system and the molecular characteristics of DNA. Carry out systematic and comprehensive research on the Tile computing model.

10.3 Programmable Tile Structure

Molecular computation can be realized by the self-assembly of the DNA Tile algorithm. Complex calculation processes require stable molecular structures and rich molecular types. Although the method of designing sticky ends with palindrome sequences can successfully construct large-scale arrays at the millimeter level, it also sacrifices the richness of molecules to a certain extent. The design of asymmetric sticky ends can not only improve the ability of specific binding, but also provide richer programming combinations. Shi Xiaolong and others from Guangzhou University [7] proposed a Sub-Tile programmable hierarchical self-assembly strategy using asymmetric sticky end design. Professor C. Montemagno, Chairman of the Canadian Intelligent Nanosystems Research, listed Sub-Tile as one of the 15 important milestones in DNA nanotechnology since its development in 1991.

Sub-Tile is composed of only three DNA single strands a, b, and c, containing primary sticky ends and secondary sticky ends. single strands a, b, and c correspond to areas a_2 and c_6, b_3 and c_5, b_1 and c_4 complement each other to form a Sub-Tile structure. Primary sticky ends such as a_1, a_3, b_4, c_1, c_2, c_3 are used to realize the connection between Sub-Tiles to form a composite structure. For example, Sub-Tile S_i and S_j are connected by complementary pairing of a_{i3} and c_{j2}, and c_{i2}, c_{i3} of S_i and b_{j4}, a_{j3} of S_j are connected to the corresponding areas of adjacent Sub-Tiles . For example, c_{i2} and a_{j3}, c_{i3} and b_{j4} are connected to form a two-arm structure, and the connection of the six-arm structure. Secondary sticky ends such as a_{i1} and c_{i1} can make the composite structure form more complex two-dimensional or three-dimensional arrays.

The asymmetric sequence design method of sticky ends lays the foundation for Sub-Tile to stably construct rich structures. On the one hand, programmable self-assembly can be achieved by simply modifying the sticky end sequence to form different structures. Experiments have proven that two-arm, three-arm, four-arm, five-arm, and six-arm structures have all been successfully constructed. Structure determines function, and the successful assembly of different types of structures also means that Sub-Tile has the potential to implement different functions. On the other hand, the asymmetric sequence design improves the sequence specificity and reduces the possibility of mismatches during the assembly process.

The assembly process of Sub-Tile adopts a layered self-assembly strategy, which reduces non-specific binding and chain interference of DNA strands during the annealing process compared to one-pot assembly. After assembling the first-

level Sub-Tile unit, the layered self-assembly strategy allows for more design combinations when binding between Sub-Tiles, forming richer assembly patterns.

10.4 Single-Strand Tile Calculation

DNA Tile utilizes the accuracy of DNA base complementary pairing and the stability of Holiday junctions to assemble various arm-like structures, but building complex shapes that can be uniquely addressed based on DNA Tile is a major challenge.

Traditional DNA Tiles are often assembled into compact structures using multiple chains, leaving a few sticky ends for further connection. Starting from first principles, the Yin Peng team at Harvard University proposed the concept of SST (single strand Tile), which uses four coding regions of a single DNA strand to achieve programmable self-assembly, and then assembles complex patterns with a large number of coded SSTs to build a DNA canvas [17]. A DNA single strand of 42 bases is divided into four binding domains, each consisting of 10-11 bases. During the assembly process, all SSTs are folded into Tiles of the same shape and size, 3 nm × 7 nm. The binding domains are divided into two groups, with domains 1 and 2 forming one group, and domains 3 and 4 forming another group. In a manner similar to "bricklaying", different SST binding domains are assembled into a plane through sequence design and angle control between adjacent helices. Theoretically, this method can be assembled into rectangles of any size. The edges of the rectangle can be maintained flat by adding DNA single strands that pair complementarily, or nanotubes can be formed by adding SSTs that connect two edge helices.

Each SST is equivalent to a pixel point. Researchers used 362 different SSTs to assemble a "molecular canvas" that is 24 helix bundles long and 28 helix bundles wide. Depending on the expected design of the figure, the idea of "paper cutting" can be used to remove unnecessary SSTs from the "molecular canvas" to form structures of different shapes. To improve

To increase the yield of structures and reduce structure aggregation, two methods were used to seal the exposed chains at the edges, both of which achieved ideal results. One method is to replace the sequence of the exposed binding domain with a poly (T) chain of the same length, and the other method is to add a protective chain. The sequence of the protective chain consists of a sequence that pairs complementarily with the exposed binding domain and a poly (T) of 10–11 bases long. Considering the experimental cost and the least number of chains, the second method was chosen to successfully construct 107 patterns, including numbers, letters, punctuation, Chinese characters, emojis, etc. The yield of constructing figures by this method is 6–40% . Experiments have shown that after mixing various different patterns that have been assembled, the patterns are independent of each other and do not interfere with each other.

The method of assembling figures using SST to construct a two-dimensional "molecular canvas" does not require careful adjustment of the stoichiometric ratio of

the added DNA chains. During the SST assembly process, nucleation is first sparse and slow, and then assembly is completed quickly. This simple and modular method effectively challenges the view that modular components (such as DNA Tiles) are not suitable for assembling complex, individually addressable shapes.

In the same year, Yin Peng and others transformed the functional perspective of SST from "pixel point" to "brick", achieving a breakthrough from two-dimensional canvas to complex three-dimensional structure [18]. Each "brick" is composed of a DNA single strand 32 bases long, with four binding domains each 8 bases long. Unlike the two-dimensional structure, the connection of SST is similar to the connection of Lego blocks, with two types of connections: one is similar to the connection of SST in the two-dimensional canvas, which is a horizontal connection, and the other is a vertical connection. In order to construct the 90° dihedral angle required for the three-dimensional structure, two SSTs only pair complementarily 8nt, that is, three-quarters of a helix. When constructing a three-dimensional structure, every 8 bp is considered a layer, and all SSTs in the layer are staggered in the same direction like "bricklaying". The layers are arranged in a 90° counterclockwise rotation, that is, every four layers form a cycle unit.

A cube structure with a length of 10 helix bundles, a width of 10 helix bundles, and a height of 80 bp ($10H \times 10H \times 80B$) is constructed as a three-dimensional molecular canvas. With an 8 bp double strand as a solid element, the three-dimensional canvas can be abstracted into different voxels. The construction of complex figures is like carving on a three-dimensional canvas.

Firstly, design the graphics. Based on a $10H \times 10H \times 80B$ three-dimensional canvas, establish the x, y, z axis spatial coordinate system. With the assistance of 3D modeling software, unnecessary voxels can be removed from the three-dimensional canvas according to the target structure, and the required SST can be selected.

Secondly, optimize the structure. After deleting some Tiles from the three-dimensional canvas, there are exposed sequences at the boundary. In order to reduce unwanted non-specific binding and improve the stability of the structure, protective chains and boundary chains can be added to optimize the structure design. The protective chain refers to the exposed 8nt sequence at the original boundary, which is replaced with 8 consecutive thymidines. The boundary chain is a 16nt half Tile that can merge with a previous 32nt complete Tile along the helical direction to form a 48nt long chain.

After sequence and structure optimization, the automatic liquid dispenser adds related DNA chains from the selected SST subset to anneal and form the structure. Successfully constructed 102 different shapes, including ellipsoids, spheres, and other cubic bodies, empty boxes, parallelograms, and other hollow structures. The yield of most structures is between a few percent and 30%.

The construction of two-dimensional and three-dimensional molecular canvases based on SST has realized the flexible construction of complex graphics. Although all SSTs involved in the construction of the molecular canvas have roughly the same configuration, their sequences are different. A large number of unique addressable DNAs have dramatically increased the cost and difficulty of the experiment.

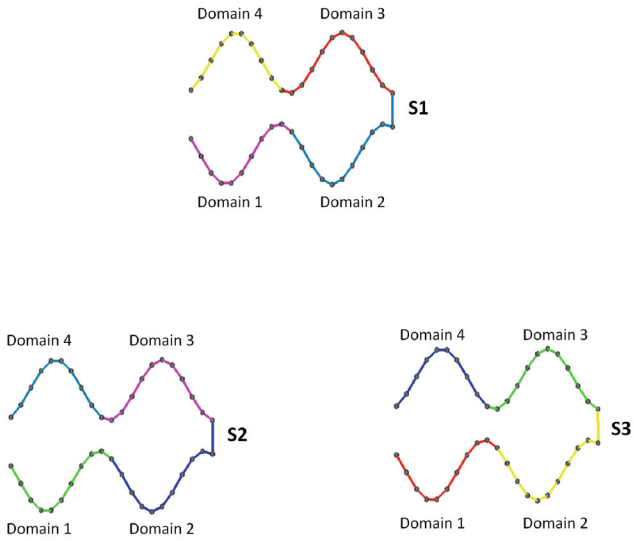


Fig. 10.7 Schematic diagram of the connection of three reusable SSTs to construct nanoscale patterns

Exploring low-cost and efficient programmable self-assembly methods has become a major challenge.

Literature [19] proposed a general method for constructing scalable nanostrips with only three reusable SSTs. The used S_1, S_2, and S_3 Tiles all have four binding domains, as shown in the Fig 10.7, parts with the same color can match each other. Two specific SSTs bind together, such as S_1 and S_2, S_2 and S_3, S_3 and S_1, the complementary pairing binding domains bind to form a helical structure. If no other auxiliary chains are added, the three SSTs will assemble freely, and due to the inherent curvature accumulation of SST, it will eventually form a DNA nanotube; if edge protection chains are added, it can prevent the two edges of the tube from complementing each other, forcing the DNA nanotube to open into a nanostrip. The edge protection chain is determined according to the sequence of binding domains 1 and 2 of S_1, dividing the binding domains 1 and 2 of S_1 into 3 parts, and the different combinations of the three parts of complementary sequences form three different protection chains, each with a length of 56 bases. In addition to controlling the shape of the structure, a strategy of the concentration ratio of the edge protection chain and the filling chain is proposed to control the length of the nanostrip. The filling chain is to close the exposed areas on both sides of the nanostrip, with 4 on each side. The proportion of nanostrips decreases as the concentration ratio between the edge protection chain and the filling chain increases.

Literature [20] proposed a scheme for constructing a larger diameter nanotube with only 2 SSTs involved, and constructed nanotubes with widths of about 100 and 300 nm. Similar to the previous SST construction nanostructure scheme, SST is divided into four binding domains. In the design scheme of the 100 nm nanotube,

the sizes of the four binding domains are 10nt, 11nt, 10nt, 11nt; the sizes of the four binding domains of the 300 nm nanotube are 9nt, 12nt, 9nt, 12nt. As shown in the figure, the SST pairing methods of the two design schemes are the same: the binding domain 1 of SST1 and the binding domain 3 of SST2 complement each other, the binding domain 2 of SST1 and the binding domain 4 of SST2 complement each other, and the binding domain 1 of SST1 and the binding domain 1 of SST2 complement each other.

Under the atomic force microscope, not only large-radius nanotubes were observed, but also star-shaped Aster structures were observed. The hierarchical assembly of 2 SSTs can explain this phenomenon: starting from the Aster structure (the nucleation point of SST), through

Adding Tiles, multiple 2-helix nanowires grow and extend; nanowire bundles extending from the same Aster structure are laterally interconnected through domain exchange to form nanotubes. The hierarchical assembly strategy bypasses the kinetic traps in the assembly process of flexible SSTs, which helps to construct wider DNA nanotubes composed of 2 SSTs.

The 2-SST strategy may pave the way for building large-scale complex supramolecular nanostructures from simple DNA assembly units to achieve multifunctional applications, reduce the manufacturing cost of nanostructures, especially in in vivo applications such as drug delivery.

The introduction of the SST strategy has dramatically increased the complexity of constructing nanostructures based on DNA Tiles. The modular and scalable method provides a controllable and programmable powerful platform for functional nanomaterials and molecular computing. The assembly of nanostructures is generally divided into two stages: the nucleation process and the growth process. However, nucleation in the SST self-assembly process is random, and different small nuclei exist in different positions of the target structure [21, 22]. In addition, although SSTs have different sequences, the same structure and connection method have resulted in low-probability events in the microscopic mechanism of SST connection, so the overall assembly is very slow. Literature [23] proposed the addition of DNA "seed chains" to trigger rapid and controlled nucleation, to promote subsequent rapid growth and improve assembly speed.

The long seed chain contains multiple binding domains and multiple SST complementary pairs, so the probability of the seed chain interacting with SST is greater, the height of the nucleation free energy barrier is reduced, and the probability of nucleation at the seed chain is greater. Secondly, computer simulations and experiments have shown that the seed assembly strategy significantly improves the optimal assembly temperature of the nanostructure, and the assembly temperature of the seed strategy has increased by about 5 °C. In traditional SST systems at lower temperatures, SST tends to bind and aggregate incorrectly, and if the nucleation barrier is low, it leads to the incorrect formation of multiple nuclei. At higher temperatures, the incorrect binding between SSTs will decrease, the nucleation barrier will be larger, and the large-scale nucleation of SST will be inhibited. The seed strategy reduces the height of the nucleation barrier, allowing SST assembly at higher temperatures, maintaining the advantage of reducing incorrect binding while

also promoting the nucleation process, and the SST largely follows the path guided by the seed during the growth stage.

10.5 Universal DNA Computer Based on SST

As we know from the previous sections, various DNA Tile self-assembly structures provide a convenient coding tool for DNA computing. Compared with other DNA computing materials, we can intuitively see the calculation results and even the entire calculation process with the help of nanoscale microscopes. This high degree of visualization is a major feature of the DNA Tile computing model.

In these models, Tiles are of various shapes, showing different DNA winding methods from the perspective of biomaterials, and different definitions of coding or functional areas from the perspective of computing models. In other words, when researchers build the functions of DNA computing models, they carefully consider how the materials used interact with each other, and how to code and execute algorithms to embed molecular self-assembly behavior into the algorithm process, so different Tiles are often only used for computing functions that match the structure itself.

The advent of SST has injected new blood into the DNA Tile family. Since its theoretical first publication in Science in 2012, the "Lego"-style splicing method of DNA single strands in the nano world has been eye-opening; later, the assembly mechanism was gradually explored, and researchers analyzed its assembly stability from the perspective of chemical reaction principles in the three axes of the Cartesian coordinate system, giving SST the ability to build like building blocks; by 2015, a universal coding strategy in 3D space was formally proposed, constructing a coding space composed of 30,000 unique SSTs. These achievements prove to the world that, due to a complete self-assembly mechanism and feasible coding strategy, this oligonucleotide chain with only 32 base lengths has the ability to serve as a universal computing material.

As we know from the introduction in reference [23], when using DX Tile to output the calculation process of the Sierpinski triangle, in order to ensure controllable input values, single-stranded DNA and DNA origami were introduced as calculation starting points, and named "seeds". Seeds are similar to a function in an electronic computer that reads for input, passing the values entered by the user to the function that performs the calculation and starts the program running. Since DNA computing occurs in parallel in liquid, the seed's transmission function is manifested in the partial sequence complementing the input sequence, used to capture the input DNA floating in the solution; the startup function is manifested in using spatial organization ability to write the input sequence into a suitable position to start the complementary reaction. DNA origami with unique addressing naturally takes on the task of seeds, and DNA origami technology will be introduced in the next section. This section uses two examples to show instances of building universal computing models using SST with the intervention of DNA origami seeds.

10.5.1 Iterative Boolean Circuit Computing Model Based on SST

This model was proposed in literature [24] in 2019, proposing the creation of a reprogrammable self-assembly system to make up for the small size of the tiles growing the Sierpinski triangle in the aforementioned algorithm. Since hundreds of Tiles are needed, SST becomes the optimal choice. The design of this biological computing model goes through the following five levels.

Design of Abstract Computing Model
The abstract computing model of the system is named Iterated Boolean Circuit (IBC). The model uses multiple layers of Boolean logic gate arrays that are executed repeatedly, with Boolean logic gates locally connected within and between layers.

In terms of composition, the IBC has two types of Boolean logic gates: logic gates and edge gates. A circuit with an input bit number of n and iteration layers of l requires $(n-1)l$ types of logic gates and $2l$ edge gates. Each logic gate has 2 inputs and 2 outputs, and each edge gate has 1 input and 1 output.

In terms of function, the system will perform calculations by repeatedly iterating through the circuit layers: starting from the input position on the left, adding copies of the circuit layer one by one, until it reaches a fixed state or goes into a loop. Since all $(n+1)l$ Boolean logic gate logic functions are specified by the user, this is a universal computing model that can be used to simulate the functions of many other computing models.

Abstract Tile Assembly Model
The second-level design abstracts Boolean logic gates into square Tiles, describing the self-assembly behavior of individual Tiles guided by the "glue" on the four edges. The glue carries the assembly information encoded by humans. Since DNA computation occurs in numerous duplicate copies present in the solution, the Tile assembly model assumes an excess of each monomer in the solution, meaning they will not be exhausted.

The relationship between the Tile model and the first-level IBC. Boolean logic gates are implemented by a single abstract square Tile, each Tile contains two or four abstract squares, representing glue, corresponding to the input/output of Boolean logic gates. The difference is that a 2-input 2-output logic gate contains four abstract squares, while a 1-input 1-output edge gate contains two abstract squares. The model assumes that Tiles start to attach and grow from the seed position representing circuit input, the formation of the assembly represents the execution of the circuit, and in principle, it continues forever.

The model also considers a cooperative assembly mechanism: to ensure the correct assembly rate, during the growth of the assembly, a Tile can only attach to the assembly when at least two types of glue match.

Since the circuit executes in the horizontal direction and the glue encodes in the vertical direction of the circuit, this design also brings an important benefit. From

the perspective of the formation of self-assembled structures, when DNA nanotubes or nanostrips form, vertical assembly can create substantial kinetic barriers to spontaneous nucleation. Therefore, suppressing spontaneous nucleation means a high assembly yield at the molecular implementation stage.

Design of Proofreading Blocks
In the third abstract level, each Tile of the second level is divided into four "proofreading blocks". There are eight glues around the Tile, two pairs correspond to one bit of input/output of the logic gate. The central glue is used for proofreading, and their positions within the proofreading block are unique.

The introduction of proofreading blocks makes matching more reliable. Only when both bits of the proofreading block match the assembly can the block stably attach. Compared with direct implementation, the addition of a proofreading mechanism reduces the matching error rate.

SST Binding Domain Design
The fourth abstract level defines the correspondence of SST binding domains, mainly considering the implementation process other than sequence design.SST logically contains four binding domains, each domain corresponds to each glue, and the proofreading block in the third level is implemented by an SST molecule.

In addition to the binding relationship, the fourth level also considers several key geometric factors during implementation:

(i) SST lattice. This includes selecting an appropriate number of bases for the binding domain.
(ii) Input adapter chain. This includes cascading the input adapter chain after DNA origami seeds to initiate the assembly of molecular logic gates.
(iii) Labeling position. This includes choosing the modification position of biotin labeling on SST for visualizing the computation process.

DNA Sequence Design
Although in the first four levels of design, the expected behavior can be described by the binding relationship of SST, the effectiveness of the abstract model in the test tube depends on the uniformity and specificity of the attachment energy between the binding domains, so the DNA sequence design of the fifth level is still challenging. This level requires that DNA strands can perform the required actions and are not prone to other actions, including:

(i) False nucleation: SST grows spontaneously without starting from the seed.
(ii) SST attachment error: Unmatched domains attach.

At the same time, there is evidence that random sequences and lightly designed sequences can undergo SST self-assembly, but the high mismatch rate is not suitable for algorithmic assembly. Therefore, in the sequence design level of this model, based on the NUPACK and ViennaRNA two nucleic acid analysis models, a temporary energy model was built to generate DNA sequences, the model includes the following considerations:

(i) Ensure the energy uniformity of SST in tiling events.
(ii) Ensure the release of binding between adjacent nanotubes during tile attachment.
(iii) Minimize the mismatch energy between the correct domain and the wrong domain to enhance specificity.

Specifically, a random local search algorithm is used in the above energy model to solve the multi-objective optimization problem, aiming to obtain a set of molecules to implement the expected interactions in the fourth-level design.

In the above iterative Boolean function computation model, SST has programmability, which is crucial for demonstrating various circuits and creating patterns of any shape. For programmability, the set of SSTs growing on origami seeds can be regarded as an "algorithmic molecular canvas", and the concept of layered design allows researchers to "carve" algorithms like Tile programmers to generate more and more deterministic algorithms for the required function.

10.5.2 Computation Model Based on Repeatable SST

The model was proposed in 2022 in literature [25], aiming to combine DNA origami structures with fewer SSTs to construct a portable and reusable DNA programmable self-assembling nanostructure service for DNA computation. To avoid SSTs falling into kinetic traps during the assembly process, the model cleverly introduces a framework-shaped DNA origami as a seed, simplifies the model with the strategy of "offset connection", and only uses two types of reusable SST units in the final model. The design and implementation of the model go through the following three levels:

1. DNA programmable self-assembling nanostructures.
   This level discusses how self-assembling nanostructures, constructed by two types of nanomaterials, work as a computational model. The nanostructure includes a framework-shaped DNA origami and an SST computational core. Therefore, the construction and operation of the model proceed in the following two steps:

   (i) Constructing DNA origami. Once the construction is completed, the restrictions for performing calculations are in place, and the calculation can begin.
   (ii) Filling in SSTs. The growth process of SSTs represents the execution of the calculation.

   After the two steps are completed, the calculation results can be read from the self-assembling nanostructure jointly generated by DNA origami and SST.
2. DNA Origami
   This level discusses how DNA origami, as a limiting factor and initiator, works, and also includes considerations for the design of DNA origami structures. DNA origami is divided into two parts:

(i) Skeleton. It is composed of eight parallel double helices, which ensure the overall stability of the self-assembling nanostructure.
(ii) Framework. It is composed of ten parallel double helices, with the first row, the last row, and the parts of DNA on the left and right columns serving as supporting edges. The supporting edges can limit the growth of SSTs and capture SST molecules from the inside, allowing SSTs to fill in from four directions simultaneously, which is used to limit and initiate reactions.

Since the skeleton and framework respectively ensure the robustness of the self-assembling structure and computational function, DNA origami can serve as a seed to pass on input values and initiate computation.

3. SST Computational Core

This level mainly discusses how SSTs perform computational functions in the origami framework, including the binding relationship of SSTs and growth restrictions. The binding relationship is shown in Fig 10.7. This model uses two types of SST structures S_1 and S_2. S_1's binding domain from the 5' to 3' direction is [a, b, c, d], S_2's binding domain is [c*, d*, a*, b*], therefore, they can alternately connect and assemble in the 2D plane. The growth of SSTs is limited by the DNA origami framework, and the restricted filling process includes the three steps.

(i) The thin supporting edge is on the outermost layer, composed of reserved scaffold chains.
(ii) The SSTs directly bound to the supporting edge are called "auxiliary chains". They have two binding domains complementary to the scaffold sequence, representing the values input by the user.
(iii) Other SSTs (S_1, S_2) bind to the binding domains of the auxiliary chains from four directions at the same time and gradually fill towards the center, representing the process and result of the computation.

In summary, this strategy of filling SSTs into a rigid nanostructure assembled within the thin edge of the origami framework, and then secondarily assembling the required shape, will greatly reduce the cost of controllable self-assembly and improve efficiency.

10.6 DNA Origami Computation

DNA origami technology is a branch of self-assembling nanotechnology. Compared with DNA Tiles, DNA origami also precisely designs nucleic acid sequences, allowing different single-stranded DNA to spontaneously complement and assemble into specific structures. The difference is that DNA origami additionally introduces a long single-stranded DNA as a scaffold to assist the assembly process.

This section first explains what DNA origami technology is, and then introduces its many researches in the field of DNA computation, including programmable self-

assembly, surface computation based on origami, and computational models where the origami itself serves as a computable structure.

10.6.1 DNA Origami Technology

DNA Origami (DNA The concept of origami was first proposed by Rothemund at the California Institute of Technology in 2006 [26]. This is a practical technique that can guide DNA strands to assemble from the bottom up to construct specific shape structures within the range of tens to hundreds of nanometers. By artificially encoding and designing hundreds of short staple chains (staple), a single-stranded circular DNA scaffold chain (scaffold) usually extracted from the M13mp18 bacteriophage of E. coli is bound and folded into a preset shape.

In recent years, the research scope of DNA origami is still expanding, the complexity of DNA origami structure morphology is constantly increasing, and the design and implementation means are also being optimized and innovated. From the development of two-dimensional symmetry at the geometric shape level to three-dimensional asymmetry [27], and then to the secondary assembly [28] and dynamic assembly [29] at the assembly method level. There are also studies on larger or smaller scale origami, and the exploration of origami with other materials (such as RNA) [30]. As well as innovations in folding methods: only scaffold chains and multiple scaffold chains [31]. With the development of DNA origami design software, design, simulation and even automatic generation tools are emerging [32], which is very user-friendly. The emergence of eye-catching results such as nanoflasks [33], DNA puzzles [34], DNA portraits [35], etc., proves the powerful ability of DNA origami to construct nanostructures.

From its design principle, it can be known that DNA origami has addressability and programmability. Addressability means that after the bottom-up splicing is over, the designer can access any specific base position on any staple on the origami through the sequence, and programmability means that the designer can easily operate the sequence to create any shape or pattern within the scaffold range. This addressability and programmability make it convenient for designers. Designers can give assembly logic and restrictions, allowing DNA origami monomers or between monomers to self-assemble according to established rules. This is the core of DNA self-assembly and the core idea of establishing a computational model based on DNA origami. The implementation results of many of the above-mentioned studies show that it also has stability in vitro, and stability provides an experimental basis for DNA origami for computational materials.

The performance of DNA origami in computational scenarios compared to DNA Tiles, by introducing scaffold chains, has improved the problem of a large number of short chains involved in Tile self-assembly, making DNA origami a larger and more stable material. In terms of computational applications, a larger scale means better information storage capacity, that is, a DNA origami unit can carry more codable addresses, which increases the diversity of models, and even large origami

can be used to "organize" smaller Tiles. These characteristics make researchers have an easy-to-use nanotool when constructing and implementing DNA computational models.

10.6.2 Programmable Self-Assembly of DNA Origami

The programmable self-assembly of DNA origami refers to viewing the origami monomer as a unit, tiling it on a plane like a Tile, or assembling it into a specific shape. Limited by the length of the universal scaffold chain 7249 bases, the size of the DNA origami monomer is limited to 50–200 nm. Therefore, by encoding a part of the base sequence on the DNA origami, allowing the origami to assemble and amplify into larger-scale or even micron-level DNA structures, is a strategy to break through the size limit.

Under normal circumstances, the programmable self-assembly between origami can be achieved through the interaction between nucleic acids, such as base complementary pairing. The "Mona Lisa" portrait in literature [35] is based on sticky end coding edges. This study uses local assembly rules and divides the self-assembly process into multiple stages. Each square represents an origami monomer. Four monomers are assembled into a 2×2 array in the first layer, and so on, four 4×4 arrays are assembled into a 8×8 array in the third layer. This layered design allows the edge codes of different arrays at the same stage to be reused, and the same scale assembly can be completed with fewer edge codes; and in principle, it reduces the number of origami participating in the reaction within a given time period, which is beneficial to reduce erroneous interactions. The layered design is also reflected in the molecular implementation process, the number of double helices involved in coding on the edge of the origami, the reaction temperature decreases layer by layer, and the reaction duration increases layer by layer. The entire self-assembly process is guaranteed by the layered design and experimental aspects, and the final product is an 8×8 scale, 0.5 square micron-sized nanoarray used as a molecular canvas, drawing portraits, roosters, circuits and other diverse patterns, intuitively showing the precise patterning at the nanoscale.

In addition to complementarity of base sequences, self-assembly can also be achieved through other forms of nucleic acid action, such as base stacking. Literature [36] uses the inherent non-sequence specificity of base stacking forces to propose a self-assembly scheme for random coding. Finally, a micrometer-scale planar DNA array with random assembly function was constructed, including the realization of random ring patterns, mazes, and trees. This shows that by making corresponding changes to the coding strategy, programmable self-assembly arrays with specific functions can also be constructed.

The programmable self-assembly of DNA origami is similar to the tiling or three-dimensional assembly of DNA Tiles, providing a structural basis for DNA computing. Because of its larger area, the two-dimensional array assembled can serve as a substrate for organizing and accommodating other DNA computing

models; it can also directly map the assembly process to logic and algorithms, allowing DNA origami itself to participate in the calculation as a computing element. These two application methods will be introduced in the following two sections.

10.6.3 DNA Origami Surface Computing

From the addressability of DNA origami, we know that its surface area of nearly ten thousand square nanometers contains about 200 addressable sites. This global addressing feature makes it an accurate positioning template or framework down to the base resolution, with obvious results in spatial organization.

In response to the problem that the scale and computing speed of traditional DNA circuits are easily constrained by the increase in the number of molecules, some theoretical studies believe that certain means can be used to divide the DNA circuit into regions to optimize the computing speed and expand the circuit scale. DNA origami, where each base on the surface can be addressed, is well suited for this task. Seelig and others at the University of Washington discussed a computational architecture of DNA strand displacement circuits in 2013 [37], proposing that spatial isolation can be achieved through the surface of two-dimensional DNA origami, making it easier to design circuits based on DNA. The Seelig team successfully moved scalable DNA logic gates and their transmission channels to a platform based on rectangular DNA origami in 2017 [38], completing the co-location of circuit elements by fixing independent hairpins on the surface of DNA origami. The arrangement and spatial organization of the origami for the hairpin logic gates make the reaction preferentially occur between "neighbors", which allows the signal transmission to proceed along the carefully arranged hairpins like wires. Compared with the molecular circuits diffused in the solution, the interactions between the circuit elements located on the surface are limited to the proximal end, which increases the reaction speed.

Like molecular circuits, molecular computing models can also achieve spatial positioning on the surface of DNA origami. In 2019, Chao Jie and others at Nanjing University of Posts and Telecommunications constructed a single-molecule DNA maze navigator [39], whose navigation principle relies on the cascade strand displacement reaction on the origami surface. After the navigation is initiated by the trigger site A, it automatically advances along the path defined by the DNA hairpin and autonomously explores one of the possible paths of the tree. In terms of application, a certain leaf node can be defined as an exit. By exploring all possible paths with a single molecule, and then using molecular purification means to separate the origami that has passed through a certain exit, the specific solution path between two points can be obtained from the collection of origami navigators. The final study defined a ten-node rooted tree on the DNA origami platform and performed depth-first search in parallel in the solution. In addition to the feature that the DNA computing process can be directly associated with biomolecules,

this research has the potential to translate biomedical sensing and decision-making problems into graphical expressions on the surface of DNA origami.

The characteristics of origami participating in the positioning of computing elements are:

(i) The interaction between circuit elements is limited to neighbors, and the overall reaction speed is faster.
(ii) Spatial positioning results in a higher effective concentration of circuit components than the effective concentration of components in a mixed solution, which helps to regulate stoichiometry.
(iii) Since the information flow no longer relies on sequence specificity control, the sequence between components is allowed to be reused.
(iv) Different functional modules can be divided in the same test tube, which is convenient for some application scenarios. Without the precise positioning and organization of DNA origami for molecular groups, these DNA computing applications are not easy to achieve.

10.6.4 Computable DNA Origami Structure

Due to its integrity, stability, and programmability, DNA origami, like DNA Tiles, can also be used to implement algorithmic operations, interact with other structures, and execute computing processes. In 2022, Xu Jin and others at Peking University [40] proposed and implemented a graph computing model based on DNA origami. This research proposes a group of DNA nano "agents" as computing units, each computing unit is composed of a DNA origami, which can be interconnected with two or more other computing units through DNA probes.

This study has realized a specific instance of the three-coloring of a graph, an NP-complete problem, as shown in Fig. 10.8. A 6-vertex graph needs to be colored with 3 colors, and the two vertices on an edge cannot be the same color. The color information of the DNA computing unit is displayed by color markers, specifically assigned to 3 different DNA origami structures, which are marked with different patterns on their surfaces for color marker recognition under a microscope. The DNA molecules used for connection are called DNA probes, which encode the edge set through the structural information of the graph, specifically manifested as several arms with sticky ends protruding from each DNA origami. According to the definition of the graph vertex coloring problem, there is no edge with the same color at both ends, so such DNA probes will not be generated when encoding the edge set. When the operation starts, under the action of the DNA probe, the DNA computing unit freely combines and connects in three-dimensional space according to the encoded graph information, finally forming a 3-color 6-vertex graph representing the calculation result. The calculation result can be directly read under an atomic force microscope. This study shows that DNA origami and DNA Tile can serve

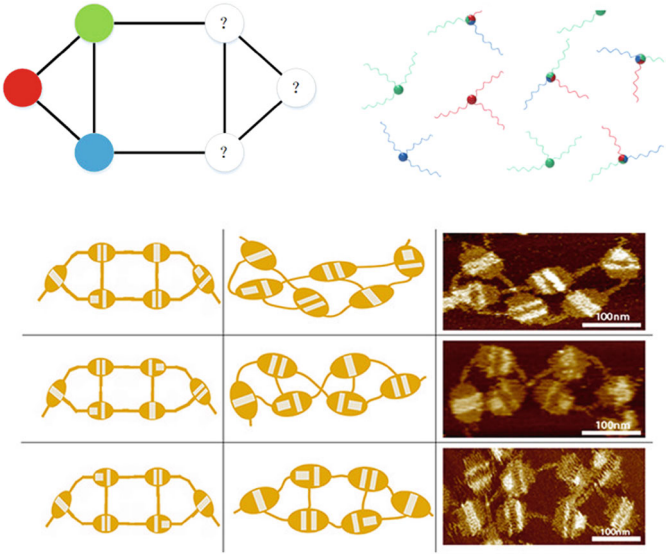


Fig. 10.8 Principle and results of the 3-vertex graph coloring problem based on DNA origami self-assembly

as computing components, performing molecular calculations and displaying the calculation process and results in self-assembled structures.

In addition to executing specific calculation instances, molecular interactions can also be used for data encryption, serving as a unique information security method. Considering the structural potential of self-assembled structures, literature [41] uses DNA origami for information steganography. This study encrypts and stores information as a continuous dot pattern in a manner similar to Braille by designing certain specific positions of a series of scaffold chains as encryption bits. During the specific encryption and decryption process, biotin-modified single-strand DNA is attached to the encryption bit of the scaffold chain to complete encryption; after adding a specific key staple chain, the scaffold chain with modifications is assembled into a complete DNA origami to complete decryption. This study attempts to generate and transmit origami ciphertext for letters A–Z and numbers 0–9, and the final decryption process, including sample processing and AFM scanning recognition, takes about 1–2 hours. Although this is longer than electronic computers, the process complies with the "CIA triad" of information security (confidentiality, integrity, availability), and provides a novel molecular solution for information encryption.

As a highly addressable, programmable nanostructure, DNA origami makes many models and engineering that precisely depend on molecular geometry and molecular dynamics possible, and also inherits and carries forward the programmable self-assembly ability of DNA Tile. The computing research based on DNA origami mentioned in this section also benefits from this, but these are just the

tip of the iceberg of its application scenarios. Some computing models that depend on the precise three-dimensional spatial position of molecules and manipulation, and the operation process relies on molecular organization, can in principle be implemented on the DNA origami platform.

References

1. Seeman, N.C.: Nucleic acid junctions and lattices. Journal of Theoretical Biology **99**(2), 237–247 (1982).
2. Fu, T.J., Seeman, N.C.: DNA double-crossover molecules. Biochemistry **32**(13), 3211–3220 (1993).
3. Zheng, J., Constantinou, P.E., Micheel, C., et al.: Two-dimensional nanoparticle arrays show the organizational power of robust DNA motifs. Nano Letters **6**(7), 1502–1504 (2006).
4. Yan, H., Park, S.H., Finkelstein, G., et al.: DNA-templated self-assembly of protein arrays and highly conductive nanowires. Science **301**(5641), 1882–1884 (2003).
5. Ma, R.-I., Kallenbach, N.R., Sheardy, R.D., et al.: Three-arm nucleic acid junctions are flexible. Nucleic Acids Research **14**(24), 9745–9753 (1986).
6. Wang, X., Seeman, N.C.: Assembly and characterization of 8-arm and 12-arm DNA branched junctions. Journal of the American Chemical Society **129**(26), 8169-8176 (2007).
7. Shi, X., Lu, W., Wang, Z., et al.: Programmable DNA tile self-assembly using a hierarchical sub-tile strategy. Nanotechnology **25**(7), 075602 (2014).
8. Liu, Y., Ke, Y., Yan, H.: Self-assembly of symmetric finite-size DNA nanoarrays. Journal of the American Chemical Society **127**(49), 17140–17141 (2005).
9. Park, S.H., Pistol, C., Ahn, S.J., et al.: Finite-size, fully addressable DNA tile lattices formed by hierarchical assembly procedures. Angewandte Chemie International Edition **118**(5), 749–753 (2006).
10. Rothemund, P.W.K., Papadakis, N., Winfree, E.: Algorithmic self-assembly of DNA Sierpinski triangles. PLoS Biology **2**(12), e424 (2004).
11. Wang, H.: Proving theorems by pattern recognition–II. Bell System Technical Journal **40**(1), 1–41 (1961).
12. Winfree, E.: Algorithmic self-assembly of DNA. Ph.D. thesis, California Institute of Technology (1998).
13. Schiefer, N., Winfree, E.: Universal computation and optimal construction in the chemical reaction network-controlled tile assembly model. In: Proceedings of the DNA Computing and Molecular Programming: 21st International Conference, DNA 21, Boston and Cambridge, MA, USA, August 17–21, 2015. pp. F. Springer (2015).
14. Winfree, E.: Algorithmic self-assembly of DNA: Theoretical motivations and 2D assembly experiments. Journal of Biomolecular Structure and Dynamics **17**(sup1), 263–270 (2000).
15. Boneh, D., Dunworth, C., Lipton, R.J., et al.: On the computational power of DNA. Discrete Applied Mathematics **71**(1–3), 79–94 (1996).
16. Brun, Y.: Self-assembly for discreet, fault-tolerant, and scalable computation on internet-sized distributed networks. Ph.D. thesis, University of Southern California (2008).
17. Wei, B., Dai, M., Yin, P.: Complex shapes self-assembled from single-stranded DNA tiles. Nature **485**(7400), 623–626 (2012).
18. Ke, Y., Ong, L.L., Shih, W.M., et al.: Three-dimensional structures self-assembled from DNA bricks. Science **338**(6111), 1177–1183 (2012).
19. Shi, X., Chen, C., Li, X., et al.: Size-controllable DNA nanoribbons assembled from three types of reusable brick single-strand DNA tiles. Soft Matter **11**(43), 8484–8492 (2015).
20. Xu, F., Wu, T., Shi, X., et al.: A study on a special DNA nanotube assembled from two single-stranded tiles. Nanotechnology **30**(11), 115602 (2019).

21. Jacobs, W.M., Reinhardt, A., Frenkel, D.: Rational design of self-assembly pathways for complex multicomponent structures. Proceedings of the National Academy of Sciences **112**(20), 6313–6318 (2015).
22. Wayment-Steele, H.K., Frenkel, D., Reinhardt, A.: Investigating the role of boundary bricks in DNA brick self-assembly. Soft Matter **13**(8), 1670–1680 (2017).
23. Zhang, Y., Reinhardt, A., Wang, P., et al.: Programming the nucleation of DNA brick self-assembly with a seeding strand. Angewandte Chemie International Edition **59**(22), 8594–8600 (2020).
24. Woods, D., Doty, D., Myhrvold, C., et al.: Diverse and robust molecular algorithms using reprogrammable DNA self-assembly. Nature **567**(7748), 366–372 (2019).
25. Chen, C., Xu, J., Ruan, L., et al.: DNA origami frame filled with two types of single-stranded tiles. Nanoscale **14**(14), 5340–5346 (2022).
26. Rothemund, P.W.: Folding DNA to create nanoscale shapes and patterns. Nature **440**(7082), 297–302 (2006).
27. Douglas, S.M., Dietz, H., Liedl, T., et al.: Self-assembly of DNA into nanoscale three-dimensional shapes. Nature **459**(7245), 414–418 (2009).
28. Chen, C., Xu, J., Shi, X.: Multiform DNA origami arrays using minimal logic control. Nanoscale **12**(28), 15066–15071 (2020).
29. Berg, W.R., Berengut, J.F., Bai, C., et al.: Light-activated assembly of DNA origami into dissipative fibrils. Angewandte Chemie **135**(51), e202314458 (2023).
30. Wang, P., Ko, S.H., Tian, C., et al.: RNA-DNA hybrid origami: folding of a long RNA single strand into complex nanostructures using short DNA helper strands. Chemical Communications **49**(48), 5462–5464 (2013).
31. Dai, K., Gong, C., Xu, Y., et al.: Single-stranded RNA origami-based epigenetic immunomodulation. Nano Letters **23**(15), 7188–7196 (2023).
32. Selnihhin, D., Andersen, E.S.: Computer-aided design of DNA origami structures. In: Computational Methods in Synthetic Biology. pp. 23–44 (2015).
33. Han, D., Pal, S., Nangreave, J., et al.: DNA origami with complex curvatures in three-dimensional space. Science **332**(6027), 342–346 (2011).
34. Rajendran, A., Endo, M., Katsuda, Y., et al.: Programmed two-dimensional self-assembly of multiple DNA origami jigsaw pieces. ACS Nano **5**(1), 665–671 (2011).
35. Tikhomirov, G., Petersen, P., Qian, L.: Fractal assembly of micrometre-scale DNA origami arrays with arbitrary patterns. Nature **552**(7683), 67–71 (2017).
36. Tikhomirov, G., Petersen, P., Qian, L.: Programmable disorder in random DNA tilings. Nature Nanotechnology **12**(3), 251–259 (2017).
37. Muscat, R.A., Strauss, K., Ceze, L., et al.: DNA-based molecular architecture with spatially localized components. In: ACM SIGARCH Computer Architecture News **41**(3), 177–188 (2013).
38. Chatterjee, G., Dalchau, N., Muscat, R.A., et al.: A spatially localized architecture for fast and modular DNA computing. Nature Nanotechnology **12**(9), 920–927 (2017).
39. Chao, J., Wang, J., Wang, F., et al.: Solving mazes with single-molecule DNA navigators. Nature Materials **18**(3), 273–279 (2019).
40. Xu, J., Chen, C., Shi, X.: Graph computation using algorithmic self-assembly of DNA molecules. ACS Synthetic Biology **11**(7), 2456–2463 (2022).
41. Zhang, Y., Wang, F., Chao, J., et al.: DNA origami cryptography for secure communication. Nature Communications **10**(1), 5469 (2019).

Chapter 11
RNA Computing

The emergence of DNA computing models naturally gave rise to the development of RNA computing. However, over the past three decades, DNA computing has advanced rapidly, while RNA computing has progressed more slowly, primarily due to the structural characteristics of RNA molecules. This chapter focuses on the computational properties of RNA molecules, RNA computing models for solving **NP**-complete problems, and related research on RNA computing in the context of logic gates and logic circuits.

11.1 Computational Characteristics of RNA Molecules

Based on DNA computation [1], RNA computation research emerged [2]. However, as of the completion of this book, there have been relatively few studies on using RNA molecules to solve **NP**-complete problems, primarily due to the following characteristics of RNA molecules.

(1) The five-carbon sugar that constitutes RNA molecules is ribose, which has a hydroxyl group (–OH) at the 2′ position. This makes RNA significantly more biologically active than the deoxyribose in DNA molecules. Additionally, the half-life of RNA in biological systems is much shorter than that of DNA molecules.
(2) Most RNA molecules in biological systems primarily exist as single strands. Under specific conditions, they often fold into more complex and diverse higher-order structures, making RNA structure prediction significantly more challenging.
(3) RNA molecules exhibit poor stability in in vitro experiments. They are highly susceptible to degradation by external nucleases and require stringent experimental conditions.
(4) The key enzyme for RNA synthesis, RNA polymerase, requires a promoter sequence and is sensitive to high temperatures. As a result, RNA molecules

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_11

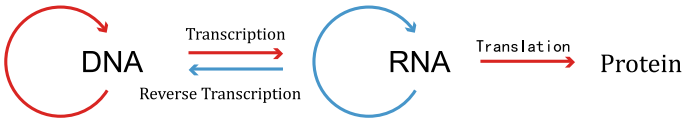


Fig. 11.1 Central dogma

cannot be amplified on a large scale using methods analogous to PCR for DNA amplification. As of the completion of this book, the cost of commonly used methods for obtaining large quantities of RNA molecules remains significantly higher than that for DNA molecules.

The aforementioned characteristics make RNA molecules less suitable for solving large-scale **NP**-complete problems in vitro. However, in the "central dogma" that describes the transmission and operation of genetic information within biological systems (see Fig. 11.1), RNA molecules play a crucial role in the three fundamental processes: DNA replication, transcription, reverse transcription, and protein translation [3]. Additionally, RNA molecules exhibit a wide variety of types and functions. Beyond messenger RNA (mRNA), which carries genetic information, transfer RNA (tRNA), which transports amino acids, and ribosomal RNA (rRNA), there are also microRNAs (miRNAs) that regulate gene expression by binding to mRNA, thereby reducing its stability or inhibiting its translation. Furthermore, small interfering RNAs (siRNAs) and short hairpin RNAs (shRNAs) are involved in post-transcriptional gene silencing, while long non-coding RNAs (lncRNAs) play significant roles in gene expression regulation, X chromosome inactivation, transcriptional repression, small molecule metabolism, and other physiological processes.

Owing to the diverse structures, physicochemical properties of RNA molecules, and their pivotal role in processing genetic information within biological systems, RNA molecules exhibit unique advantages in computational applications. By integrating the principles of biological computing with the mechanisms of information storage and processing in living organisms, they enable logical regulation and processing of intracellular information.

11.2 RNA Computation Model for Solving NP-Complete Problems

In 2000, evolutionary biologist Laura Landweber and others at Princeton University [2] first introduced RNA molecules into DNA computation research and realized a biological computation solution for the 3×3 "knight problem" on the chessboard. This problem is to find a set of position configurations where no one piece can attack other pieces, which can also be described as solving the following specific SAT satisfiability problem:

$$((\neg \mathbf{h} \wedge \neg \mathbf{f}) \vee \neg \mathbf{a}) \wedge ((\neg \mathbf{g} \wedge \neg \mathbf{i}) \vee \neg \mathbf{b}) \wedge ((\neg \mathbf{d} \wedge \neg \mathbf{h}) \vee \neg \mathbf{c}) \wedge ((\neg \mathbf{c} \wedge \neg \mathbf{i}) \vee \neg \mathbf{d}) \wedge ((\neg \mathbf{a} \wedge \neg \mathbf{g}) \vee \neg \mathbf{f})$$

Where $a, b, c, d, e, f, g, h,$ and i represent the positions on the chessboard, "true" means there is a knight on the position, and "false" means there is no knight. This model uses different sequences of DNA molecules to represent the state of a specific position on the chessboard, that is, there is a "knight" or "blank" on this position. The model first synthesizes a total of 1024 10-bit DNA chains, representing a DNA data pool that includes all possible chessboard configurations. The structure of its chain is as follows:

$5'$ prefix (24nt)-position a (value 0 or 1, 15nt)-interval (5nt)-position b (value 0 or 1, 15nt)-interval (5nt)-position j (value 0 or 1, 15nt)-suffix (32nt) $3'$

The model then reverse transcribes the synthesized DNA data pool to obtain the required RNA data pool; it cleverly uses ribonucleic acid H to specifically cut RNA molecules in the DNA-RNA double strand, thereby removing the specified RNA fragments from the data pool.

To determine the satisfaction of $((\neg \mathbf{h} \wedge \neg \mathbf{f}) \vee \neg \mathbf{a})$ as an example, the algorithm steps are as follows:

(1) Starting from position a, perform the OR operation and divide the RNA data pool into two equal parts. Add the DNA chain sequence representing $a = 0$ to one part. After renaturation, use ribonucleic acid H to disrupt the condition of $a = 0$, thus accomplishing the determination of $a = 1$. Similarly, disrupt the chains where there are knights at positions h and f (i.e., $h = 1$ and $f = 1$).
(2) Add the DNA chain sequence representing $a = 1$ to the other part. After renaturation, use ribonucleic acid H to disrupt the condition of $a = 1$, completing the determination of $a = 0$.
(3) Collect the RNA molecules from the above two parts that have not been destroyed by ribonucleic acid, purify and amplify them, and then mix them. At this time, the RNA library meets $((\neg \mathbf{h} \wedge \neg \mathbf{f}) \vee \neg \mathbf{a})$.
(4) After completing the above steps, proceed to step 2 and start the relevant screening operations from position b (see Fig. 11.2).

Finally, after multiple rounds of screening, all RNA fragments without solutions are removed from the data pool. The remaining full-length chains are gel-separated and recovered through reverse transcription and PCR, obtaining the correct answer. The schematic diagram of the algorithm is shown in Fig. 11.2. The final result diagram is obtained through gel electrophoresis, and one of the correct answers, corresponding to the values at positions a to i, is 001011000.

The RNA computing model described in this study still necessitates the involvement of DNA. Specifically, the required RNA database is obtained by first constructing a DNA library and then reverse transcribing it. Moreover, the core of the calculation–ribonucleic acid H targets the RNA strand within the DNA-RNA heterologous duplex and is classified as a non-sequence-specific endonuclease.

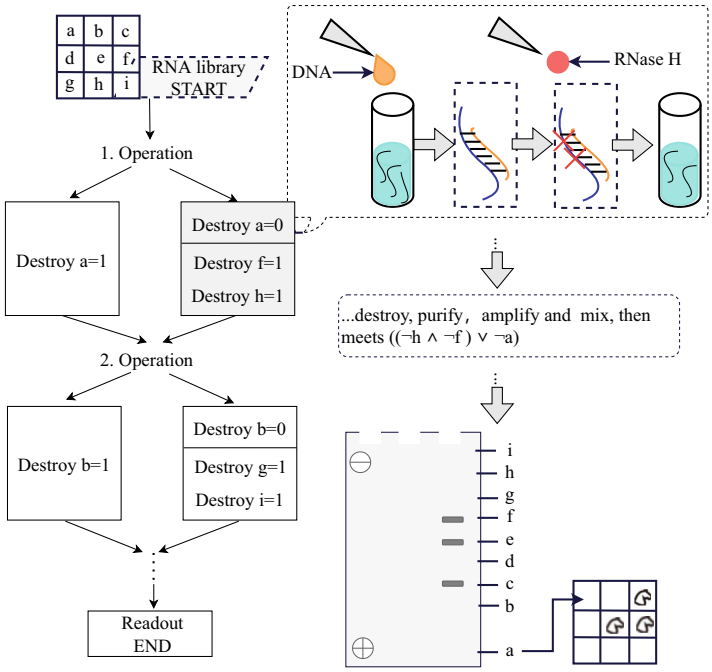


Fig. 11.2 Biological computing algorithm and result reading of the "Knight's Problem"

11.3 Related Research on RNA Computing in Logic Gates and Logic Circuits

With the rise of DNA logic gate research, logic gates involving RNA molecules are also constructed from basic logic gates, and are expected to be applied to gene expression regulation, disease detection and treatment research. This section introduces RNA sequence prediction, regulation of mRNA molecules, combination of interfering RNA and nucleic acid aptamers, and combination of gene editing techniques.With the rise of DNA logic gate research, logic gates involving RNA molecules are also constructed from basic logic gates, and are expected to be applied to gene expression regulation, disease detection and treatment research. This section introduces RNA sequence prediction, regulation of mRNA molecules, combination of interfering RNA and nucleic acid aptamers, and combination of gene editing techniques.

11.3.1 Prediction and Design of RNA Molecular Structure

In RNA computing, studying RNA molecular structures is crucial. Currently, experimental techniques mainly involve X-ray crystallography, nuclear magnetic resonance spectroscopy, and cryo-electron microscopy, which have verified numerous RNA 3D shapes [4, 5]. But for RNAs with longer sequences or complex structures, these methods are often time-consuming and labor-intensive [6], resulting in the known number of RNA 3D structures being far fewer than that of RNA sequences. By 2023, the Protein Data Bank (PDB) [7] had archived 188,726 protein 3D structures, among which only 1732 were unique RNA 3D structures, merely about 1% of the total. Meanwhile, the number of RNA sequences in the RNA Central Database [8] had reached 34 million during the same period.

The methods for RNA structure prediction have evolved from early approaches that sought to maximize base pairing and were based on thermodynamic stability, to utilizing multiple sequence alignments in bioinformatics, as well as homology modeling predictions using known RNA sequences as templates. In recent years, deep learning methods have also been applied to the prediction of RNA three-dimensional structures. Following the remarkable success of AlphaFold 2 in protein structure prediction, similar models like trRosettaRNA and ARES have also been proposed [9, 10]. Judging from the international competitions for RNA three-dimensional structure prediction, namely RNA-Puzzles and CASP-RNA (Critical Assessment of Structure Prediction—RNA) [11, 12], the existing prediction models can generally be categorized into physics-based models [13, 14], knowledge-based fragment assembly [15, 16], and deep learning-based models [9, 10, 17, 18].

In practice, in order to exert the characteristics and functions of RNA molecules, it is usually necessary to fold them into specific three-dimensional structures. As early as 1998, Zhang et al. first redesigned the prohead RNA (pRNA) derived from the $\Phi 29$ bacteriophage packaging motor and assembled it into polymeric RNA nanoparticles through hand-in-hand interactions, as shown in Fig. 11.3 [19]. Subsequently, Shu et al. invented a self-annealing technique in 2004, creating pRNA nanoparticles using palindromic sequences [20]. Subsequent research has successively proven that pRNA molecules can serve as multivalent carriers to deliver various functional molecules including aptamers, siRNA, ribozymes, and microRNA [21–25] [see Fig. 11.3b]. Currently, the modification of RNA molecules and the design of RNA nanoparticles have become an important research field in modern science [26], providing new research and treatment approaches for the computing and medical fields.

11.3.2 RNA Computing Based on Molecular Automata

In the traditional biological "central dogma", mRNA serves as the information bridge from DNA to protein, playing a crucial role in gene expression. Detecting

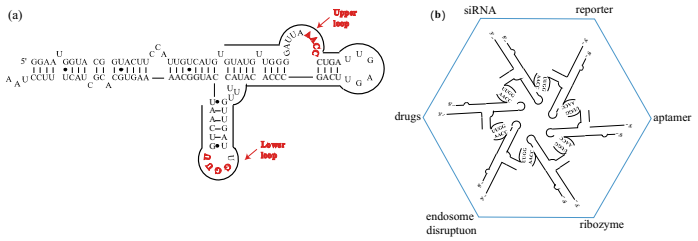


Fig. 11.3 Schematic diagram of pRNA structure and function. (**a**) Secondary structure of wild-type pRNA containing 120 bases, (**b**) Secondary structure of pRNA hexamer, and schematic diagram of carrying various components as a carrier

mRNA copy numbers and modulating the stability of mRNA molecules are fundamental strategies for treating gene-related diseases. In 2001, Benenson et al. proposed a finite-state molecular automaton model, which laid the groundwork for this approach [27]. This automaton consists of only two states, $S0$ and $S1$, and an alphabet comprising two letters, a and b, with eight possible transition rules. Both the states and letters are represented by DNA oligonucleotide sequences. The DNA chain used in the study (approximately 300 bp) has the following basic structure:

5′-spacer-enzyme cutting site sequence-spacer-state a+state b-spacer-stop state sequence-spacer-3′

After the plasmid (pBlueScript II SK(+)) is cut by an enzyme, this sequence is obtained by PCR amplification. The model employs the nucleic acid endonuclease Fok I as the hardware molecule of the molecular automaton. This enzyme specifically recognizes the sequence 5′-GGATG and cuts the DNA double strand at the 9th base on the downstream strand and at the 13th base on the complementary strand. After a series of operations including input, hybridization, enzyme cutting, and detection, its state transition is completed.

Building on the aforementioned research, the Shapiro research group focused on the post-transcriptional regulatory role of mRNA molecules in genetic information and proposed a molecular automaton model for the logical regulation of gene expression in 2004 [28], thereby linking biological computation with disease detection or treatment. This model primarily consists of three programmable modules: a computing module, an input module, and an output module. The core computing module utilizes nucleic acid endonucleases as hardware molecules (e.g., Fok I), with a DNA double strand containing the endonuclease recognition site, a spacer sequence, and sticky ends serving as the software molecule, and a specific DNA sequence as the state and alphabet set. Initially, the input molecule reacts with the hardware-software complex to form a hardware-software-input complex. Subsequently, the hardware molecule performs actions according to predefined rules. After a series of computational steps, the output consists of DNA molecules in either a “yes” or “no” state. This enables the model to automatically detect specific mRNA concentrations in the test tube based on the programmed logical rules and

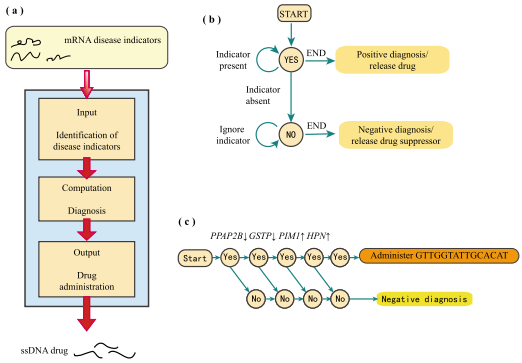


Fig. 11.4 Logical design and operation of molecular automata. (**a**) Function and module structure of molecular automata. (**b**) State transition diagram of the diagnostic automaton. (**c**) Computational steps for diagnosing prostate cancer

to generate molecules that inhibit gene expression (or release drugs) according to the detection results. This approach can be applied as antisense therapy in gene treatment (see Fig. 11.4).

The research group also conducted related in vitro experiments to simulate the feasibility of integrating molecular automata with disease detection and treatment for genes associated with small cell lung cancer and prostate cancer models [29]. For instance, in this study, the expression of prostate cancer-related genes is represented by the following symbolic rules:

$$PPAP2B\downarrow GSTP1\downarrow PIM1\uparrow HPN\uparrow$$

For each symbol, the automaton has three transitions: positive (Yes $\rightarrow$Yes), negative (Yes $\rightarrow$No), and neutral (No $\rightarrow$No) (see Fig. 11.4b). Figure 11.4c illustrates the computational steps of the automaton processing this rule. If the computation result is "positive", it indicates that the genes *PPAP2B* and *GSTP1* are underexpressed, while the genes *PIM1* and *HPN* are overexpressed. Consequently, the automaton further outputs therapeutic molecules for prostate cancer, such as the ssDNA molecule GTTGGTATTGCACAT.

11.3.3 RNA Computing Combined with RNA Interference Technology (RNAi)

RNA interference (RNAi) technology is one of the key methods for modulating mRNA expression. Among its components, small interfering RNA (siRNA), which plays a pivotal role in the RNAi process, is a small RNA molecule found in organisms, typically 20–25 nucleotides in length. siRNA guides the RNA-induced silencing complex (RISC) to recognize and degrade target mRNA molecules, thereby silencing specific genes. In 2007, the Benenson research group integrated biological computation with siRNA technology, leveraging the regulatory influence of untranslated regions on downstream gene transcription and the linear tandem

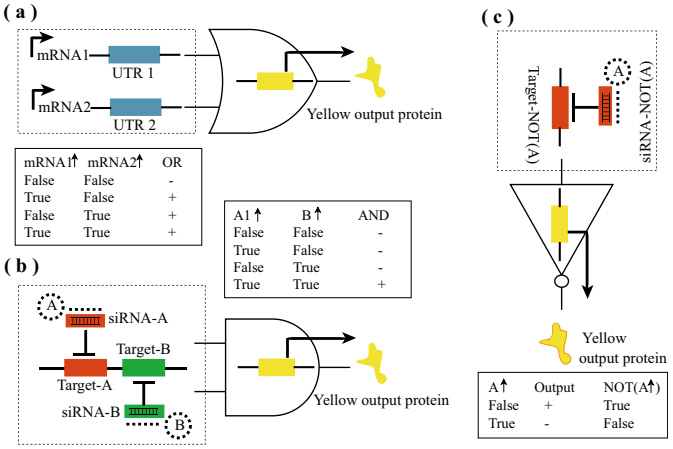


Fig. 11.5 Schematic diagram of siRNA-related logic gates. (**a**) OR gate, (**b**) AND gate, (**c**) NOT gate

arrangement of vector sequences. They constructed basic logic gates such as AND, OR, and NOT, with the optical signals generated by fluorescent proteins serving as the output after logical operations [30]. Below is a brief introduction to the basic logic gates developed in this study.

(1) OR gate: As shown in Fig. 11.5a, the fluorescent protein gene is connected in series with the mRNA1 and mRNA2 genes. As long as either mRNA1 or mRNA2 is normally transcribed and expressed, a recognition signal (fluorescent protein output) will be generated, thereby establishing an OR relationship.
(2) AND gate: As shown in Fig. 11.5b, the fluorescent protein gene, target-A, and target-B sequences are arranged in series. siRNA-A and siRNA-B can target and affect target-A and target-B, respectively. These effects can be suppressed by two endogenous substances, A and B. Therefore, only when both A and B are present simultaneously can the RNAi effect be completely inhibited, allowing the fluorescent protein gene to be expressed normally. In this way, A and B establish an AND relationship.
(3) NOT gate: As shown in Fig. 11.5c, if the endogenous substance A activates siRNA-NOT(A), then upon the addition of A, the resulting siRNA-NOT(A) will inhibit the expression of target-NOT(A). This inhibition directly leads to a reduction in the fluorescent protein product.

Based on the principles of the logic gates designed above, as long as the relationships between endogenous substances A, B, C, E and their corresponding siRNAs are predetermined, the molecular automaton can automatically perform logical judgments based on the linear arrangement of genes and output the final results when A, B, C, E, etc., are present in the system. These simple logic gates can be combined to achieve complex logical operations, such as (A AND C AND E)

or (NOT (A) AND B). The research group validated these logic gates in artificially cultured kidney cells and proposed a multi-layer logic gate construction strategy. This strategy involves regulating primary protein products first, which then control the expression of secondary fluorescent protein genes.

Another class of small non-coding RNAs, approximately 22 nucleotides in length, known as microRNAs (miRNAs), also plays a crucial role in the RNAi process. miRNAs regulate gene expression by complementary pairing with the 3′ untranslated regions (3′ UTRs) of their target mRNA molecules, leading to the degradation or translational repression of these mRNAs. Moreover, the expression of miRNAs undergoes significant changes in certain diseases, such as cancer, making them valuable diagnostic biomarkers in clinical medicine. In 2017, Xing et al. utilized an improved hairpin stacking circuit (HSC) technology to design logical computation models (AND, INHIBIT) for the precise detection of specific miRNAs, such as *miR-21* and *miR-155* [31]. Concurrently, AND and OR gates for detecting input miRNA molecules were proposed. These gates leverage the activation of luminescent states in organic small molecules, facilitating the cascading of basic logic gates into more complex logic circuits [32]. Subsequently, by integrating gold nanoparticle self-assembly, hybridization chain reaction (HCR), and DNA strand displacement technologies, logic gates for detecting cancer-related miRNAs (e.g., *miR-21*, *miR-200c*, and *miR-605*) were successively developed [33–35].

In addition, by fully leveraging the information processing mechanisms inherent in the regulation of gene expression within organisms, and integrating the continuous development of new technologies and methods in molecular biology and related fields, researchers aim to construct complex biological circuits based on fundamental logic gates. This represents one of the primary research directions in RNA-mediated biological computation.

11.3.4 RNA Computation Combining Ribozyme and Aptamers Technology

Ribozymes are RNA molecules with catalytic activity, capable of catalyzing specific biochemical reactions (such as RNA cleavage, ligation, or modification). Aptamers are special single-stranded DNA or RNA molecules that can bind to specific target molecules (such as proteins, small molecules, or metal ions) with high affinity and specificity. These aptamers are typically obtained from random nucleotide sequence libraries using the Systematic Evolution of Ligands by Exponential Enrichment (SELEX) technology [36, 37]. In 2008, Christina et al. utilized the ability of hammerhead ribozymes (HHRzs) to cleave substrate RNA at specific positions (see Fig. 11.6a) to construct an RNA-based computational device for processing intracellular information [38]. This study emphasized the modular design, scalability, and integration concepts in RNA computation. The device consists of three components: a sensor module composed of RNA aptamers, an activator module composed of

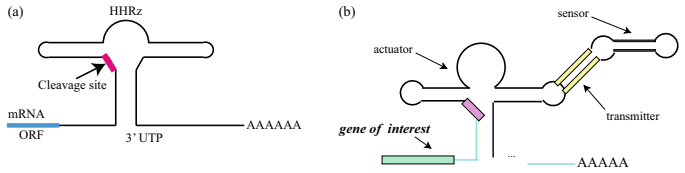


Fig. 11.6 Schematic diagram of the hammerhead ribozyme RNA computing device. (**a**) The function of the hammerhead ribozyme (HHRzs). (**b**) The functional combination framework for assembling RNA devices from modular components

hammerhead ribozymes, and an emitter module formed by coupling parts of their sequences. When the sensor module is present as input, the activated RNA device binds to the 3′ UTR of the target gene, and the ribozyme self-cleaves to inactivate the transcript, thereby reducing gene expression, as illustrated in Fig. 11.6b. Fluorescent RNA aptamers can bind to small dye molecules, stabilizing them from a flexible, rotating state into a rigid, planar conformation, resulting in intense fluorescence. This represents a novel technology for labeling and imaging RNA in live cells. In 2017, Khalid et al. employed fluorescent RNA aptamer technology to design a split-Broccoli aptamer system, implementing an AND logic gate for detecting RNA-related events within cells [39].

11.3.5 RNA Computation Combining CRISPR/Cas Gene Editing Technology

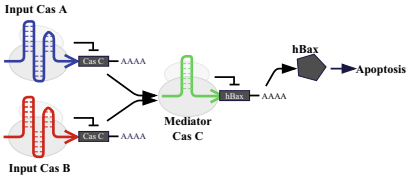
With the further development of gene editing technology, the novel RNA-protein interaction system—CRISPR/Cas system—has garnered widespread attention. This system, an acquired immune mechanism found in bacteria and archaea, consists of clustered regularly interspaced short palindromic repeats (CRISPR) and CRISPR-associated (Cas) proteins. It can recognize and cleave exogenous viral or plasmid nucleic acids, thereby protecting against the invasion of foreign genetic material [40]. Taking the CRISPR/Cas9 system as an example, its fundamental mechanism is as follows: First, the CRISPR sequence, as the recognition component, is transcribed into precursor crRNA (pre-crRNA). The pre-crRNA is then processed into mature crRNA, which possesses target recognition capability. The cleavage component, Cas9, is a nucleic acid endonuclease that forms an RNA-protein complex with the mature crRNA. As the crRNA specifically recognizes and binds to a target DNA sequence, Cas cleaves the DNA strand, creating a break in the gene. Finally, gene repair is achieved through homology-directed repair (HDR) or non-homologous end joining (NHEJ), accomplishing the goal of gene editing. Currently, the widely used CRISPR/Cas9 system recognizes double-stranded DNA (dsDNA), while the CRISPR/Cas12a system recognizes both dsDNA and single-stranded DNA (ssDNA) and exhibits non-specific collateral cleavage activity

against ssDNA. The CRISPR/Cas13a system targets single-stranded RNA (ssRNA), and the CRISPR/Cas14a system exclusively recognizes ssDNA. Except for the CRISPR/Cas9 system, all other systems exhibit collateral cleavage activity, albeit with different substrates.

The novel RNA-protein interaction system, such as CRISPR/Cas, can be directly mapped to logic switches. It has not only garnered significant attention in DNA logic gate research but has also become a focal point in the field of RNA computing. In 2019, Breanna et al. investigated several specific endoribonucleases derived from the CRISPR/Cas13 and CRISPR/Cas6 systems as effectors for RNA cleavage regulation. They selected nine endoribonucleases with the highest orthogonality for RNA activation (ON switches) and repression (OFF switches). This study successfully designed complex circuits, such as feedforward loops, single-node positive feedback and inhibition, and bistable switches. It also highlighted that such switches, with appropriate modifications, can be adapted to various RNA cleavage effectors, including endonucleases, miRNAs, and nucleases [41].

Kawasaki et al. developed the CARTRIDGE (Cas-Responsive Translational Regulation Integratable into Diverse Gene Control) method, which repurposes Cas proteins as translation repressors and activators in mammalian cells. This approach first constructs a translation OFF switch responsive to Cas proteins by inserting small guide RNA (sgRNA) into the 5′ untranslated region (UTR) of mRNA. Subsequently, it utilizes the nonsense-mediated mRNA degradation (NMD) pathway to recognize and degrade mRNAs containing premature termination codons (PTCs) within the open reading frame (ORF), thereby controlling mRNA degradation and converting the OFF switch into an ON switch responsive to Cas proteins [42].

Taking the AND gate as an example, Andreth et al. constructed a plasmid containing Cas A (PspCas13b) and Cas B (SaCas9). When both are present, an intermediate plasmid (AkCas12b) is produced. This plasmid is transfected into human embryonic kidney cells (HEK-293FT), inducing the expression of the pro-apoptotic protein hBax, thereby triggering cell apoptosis, as shown in Fig. 11.7. The study further designed 24 different orthogonal regulators (13 OFF and 11 ON for each switch) and demonstrated the construction of translational logic gates and complex logic circuits using 60 different combinations of Cas proteins.



Input Cas A	Input Cas B	Mediator Cas C	Output	Cell fate
PspCas13b	SaCas9	PspCas13b	hBax	
PspCas13b	SaCas9	AkCas12b		
0	0	1	0	Live
0	1	1	0	Live
1	0	1	0	Live
1	1	0	1	Dead

Fig. 11.7 Schematic diagram of the apoptosis AND gate and its truth table [42]. Kawasaki, S., Ono, H., Hirosawa, M. et al. Programmable mammalian translational modulators by CRISPR-associated proteins. Nat Commun 14, 2243 (2023). https://doi.org/10.1038/s41467-023-37540-7. (**a**) Apoptosis AND gate. (**b**) Truth table

The target recognition of the CRISPR/Cas system strictly follows the Watson-Crick base pairing principle, enabling effective coupling with functional nucleic acids and significantly expanding the system's application scope. This has made it a major research focus in interdisciplinary fields in recent years [43]. However, since this area of research involves multiple technical disciplines and is not confined to the field of RNA computing, the related research progress will not be further elaborated in this book.

11.3.6 RNA Computing Combined with Synthetic Biology Techniques

In 2000, researchers such as Gardner and Elowitz utilized genetic components to create a "bistable gene switch", "biological oscillator", and "logic circuit" in *E. coli*, marking the birth of synthetic biology [44, 45]. Their research concepts and ideas have mutually informed and complemented biological computing. In 2012, Ausländer et al. employed transcription switches to receive and process input signals (erythromycin and phloroglucinol) to implement highly complex circuits. These input signals utilized RNA-binding proteins (RBPs), L7Ae and MS2CP, as inhibitors of downstream translational switches [46]. Specifically, the L7Ae protein recognizes and binds to specific structures (e.g., K-turn) in RNA molecules, thereby influencing RNA structure and function. MS2 is a functional RNA-binding protein that can establish specific binding sites on mRNA. Both are essential tools widely used in molecular biology and synthetic biology. In 2015, Wroblewska et al. also leveraged these two RNA-binding proteins (MS2-CNOT7 and L7Ae) to achieve cross-inhibition, enabling bidirectional signaling pathways and feedback regulation [47].

During this period, Peng Yin et al. first proposed the concept of ribocomputing [48]. In 2017, by integrating synthetic biology regulatory elements, they utilized Small Transcription Activating RNAs (STARs) and "toehold switch" technology to construct multiple AND, OR, and NOT logic gates, enabling devices to sense and compute complex signals in living cells, as demonstrated in *E. coli*. Figure 11.8 illustrates the schematic design of ribocomputing. The device uses RNA molecules as input signals and proteins as output signals; signal processing is mediated by gate-controlled RNAs within co-localized sensing and output modules. Input molecules and gate-controlled RNAs self-assemble to form AND, OR, and NOT logic gates. When the trigger RNA binds to the complementary region on the toehold switch, the ribosome binding site (RBS) and start codon (typically AUG) of the toehold switch RNA are exposed, thereby activating translation. The optimized toehold switch retains a weak hairpin structure when activated by the trigger RNA, allowing binding to a second signal. When both signals are present, efficient ribosome translation is enabled, completing the AND gate logic operation. This study designed a nucleic acid computing system entirely from RNA molecules, offering

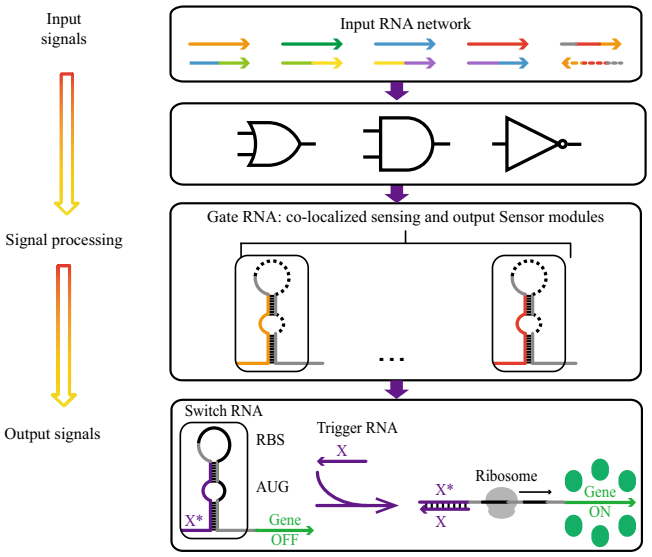


Fig. 11.8 Schematic diagram of synthetic ribocomputing devices for in vivo computation

predictability and designability. It not only reduces signal loss caused by diffusion but also enhances circuit reliability and lowers metabolic costs. Following this work, Matsuura et al. constructed an RNA-based logic circuit in 2018, comprising five 2-input logic gates and one 3-input AND gate, which uses RNA-binding proteins (RBPs) to detect multiple miRNA inputs in mammalian cells and regulate output protein expression [49].

From RNA computing for the "Knight's Problem" to in vitro molecular automata, and further extending to the construction of logic gate operations and complex circuits within cells, the representative works discussed above highlight that research in RNA computing within the field of biological computing is closely centered on the unique characteristics of RNA molecules and the latest theories and technologies from related disciplines. The research focus has transitioned from initially solving complex **NP**-complete problems to integrating the principles of biological computing with the mechanisms of information storage and processing in biological organisms. This integration aims to design novel logic operation methods and achieve logical regulation and processing of intracellular information. As of the completion of this book, the field of RNA computing has become a deeply interdisciplinary research area, combining life sciences, computer and information science, and nanotechnology materials science. It is poised to play an unprecedented and significant role in disease diagnosis and treatment, precision medicine, and information science.

References

1. Adleman, L.M.: Molecular computation of solutions to combinatorial problems. Science **266**(11), 1021–1024 (1994).
2. Faulhammer, D., Cukras, A.R., Lipton, R.J., et al.: Molecular computation: RNA solutions to chess problems. Proceedings of the National Academy of Sciences **97**(4), 1385–1389 (2000).
3. Crick, F.: Central dogma of molecular biology. Nature **227**, 561–563 (1970).
4. Kavita, K., Breaker, R.R.: Discovering riboswitches: The past and the future. Trends in Biochemical Sciences **48**, 119–141 (2023).
5. Rose, P.W., Prlić, A., Altunkaya, A., et al.: The RCSB Protein Data Bank: Integrative view of protein, gene and 3D structural information. Nucleic Acids Research **45**(D1), D271-D281 (2017).
6. Schlick, T., Pyle, A.M.: Opportunities and challenges in RNA structural modeling and design. Biophysical Journal **113**, 225–234 (2017).
7. Berman, H. M., Westbrook, J., Feng, Z., etal.: The Protein Data Bank. Nucleic acids Research **28**(1), 235–242 (2000)
8. RNAcentral Consortium: RNAcentral 2021: Secondary structure integration, improved sequence search and new member databases. Nucleic Acids Research **49**(D1), D212-D220 (2021).
9. Wang, W., Feng, C., Han, R., et al.: trRosettaRNA: Automated prediction of RNA 3D structure with transformer network. Nature Communications **14**, 7266 (2023).
10. Townshend, R.J.L., Eismann, S., Watkins, A.M., et al.: Geometric deep learning of RNA structure. Science **373**(6558), 1047–1051 (2021).
11. Cruz, J.A., Blanchet, M.F., Boniecki, M., et al.: RNA-Puzzles: A CASP-like evaluation of RNA three-dimensional structure prediction. RNA **18**, 610–625 (2012).
12. Kryshtafovych, A., Antczak, M., Szachniuk, M., et al.: New prediction categories in CASP15. Proteins **91**, 1550–1557 (2023).
13. Malhotra, A., Tan, R., Harvey, S.: Modeling large RNAs and ribonucleoprotein particles using molecular mechanics techniques. Biophysical Journal **66**, 1777–1795 (1994).
14. Wang, X., Tan, Y., Yu, S., et al.: Predicting 3D structures and stabilities for complex RNA pseudoknots in ion solutions. Biophysical Journal **122**, 1503–1516 (2023).
15. Parisien, M., Major, F.: The MC-Fold and MC-Sym pipeline infers RNA structure from sequence data. Nature **452**, 51–55 (2008).
16. Zhou, L., Wang, X., Yu, S., et al.: FebRNA: An automated fragment-ensemble-based model for building RNA 3D structures. Biophysical Journal **121**, 3381–3392 (2022).
17. Shen, T., Hu, Z., Peng, Z., et al.: E2Efold-3D: End-to-end deep learning method for accurate de novo RNA 3D structure prediction. arXiv **2207**, 01586 (2022).
18. Sha, C., Wang, J., Dokholyan, N.V.: Predicting 3D RNA structure from solely the nucleotide sequence using Euclidean distance neural networks. Biophysical Journal **122**, 444A (2023).
19. Zhang, F., Lemieux, S., Wu, X., et al.: Function of hexameric RNA in packaging of bacteriophage $\Phi 29$ DNA in vitro. Molecular Cell **2**, 141–147 (1998).
20. Shu, D., Moll, W.D., Deng, Z., et al.: Bottom-up assembly of RNA arrays and superstructures as potential parts in nanotechnology. Nano Letters **4**, 1717–1723 (2004).
21. Guo, S., Tschammer, N., Mohammed, S., et al.: Specific delivery of therapeutic RNAs to cancer cells via the dimerization mechanism of phi29 motor pRNA. Human Gene Therapy **16**, 1097–1109 (2005).
22. Shu, Y., Cinier, M., Shu, D., et al.: Assembly of multifunctional phi29 pRNA nanoparticles for specific delivery of siRNA and other therapeutics to targeted cells. Methods **54**, 204–214 (2011).
23. Haque, F., Shu, D., Shu, Y., et al.: Ultrastable synergistic tetravalent RNA nanoparticles for targeting to cancers. Nano Today **7**, 245–257 (2012).
24. Ye, X., Hemida, M., Zhang, H., et al.: Current advances in Phi29 pRNA biology and its application in drug delivery. Wiley Interdisciplinary Reviews: RNA **3**, 469–481 (2012).

25. Qiu, M., Khisamutdinov, E., Zhao, Z., et al.: RNA nanotechnology for computer design and in vivo computation. Philosophical Transactions of the Royal Society A: Mathematical, Physical, and Engineering Sciences **371**(2000), 20120310 (2013).
26. Jasinski, D., Haque, F., Binzel, D.W., et al.: Advancement of the emerging field of RNA nanotechnology. ACS Nano **11**(2), 1142–1164 (2017).
27. Benenson, Y., Paz-Elizur, T., Adar, R., et al.: Programmable and autonomous computing machine made of biomolecules. Nature **414**, 430–434 (2001).
28. Benenson, Y., Shapiro, E.: Molecular computing machines. In Dekker Encyclopedia of Nanoscience and Nanotechnology, James, A.S., Cristian, C., Karol, P., Eds.; Marcel Dekker, Inc.: New York, 2004: 2043–2056.
29. Benenson, Y., Gil, B., Ben-Dor, U., et al.: An autonomous molecular computer for logical control of gene expression. Nature **429**, 423–429 (2004).
30. Rinaudo, K., Bleris, L., Maddamsetti, R., et al.: A universal RNAi-based logic evaluator that operates in mammalian cells. Nature Biotechnology **25**(7), 795–801 (2007).
31. Xing, Y., Li, X., Yuan, T., et al.: Engineering high-performance hairpin stacking circuits for logic gate operation and highly sensitive biosensing assay of microRNA. Analyst **142**(24), 4834–4842 (2017).
32. Morihiro, K., Ankenbruck, N., Lukasak, B., et al.: Small molecule release and activation through DNA computing. Journal of the American Chemical Society **139**(39), 13909–13915 (2017).
33. Yu, S., Wang, Y., Jiang, L., et al.: Cascade amplification mediated in situ hot-spot assembly for microRNA detection and molecular logic gate operations. Analytical Chemistry **90**(7), 4544–4551 (2018).
34. Ma, X., Gao, L., Tang, Y., et al.: Gold nanoparticles-based DNA logic gate for miRNA inputs analysis coupling strand displacement reaction and hybridization chain reaction. Particle and Particle Systems Characterization **35**(2), 1700326 (2018).
35. Ran, X., Wang, Z., Ju, E., et al.: An intelligent 1:2 demultiplexer as an intracellular theranostic device based on DNA/Ag clusters gated nanovehicles. Nanotechnology **29**(6), 065501 (2018).
36. Ellington, A.D., Szostak, J.W.: In vitro selection of RNA molecules that bind specific ligands. Nature **346**(6287), 818–822 (1990).
37. Doug, I., Craig, T., Larry, G., et al.: Selexion: Systematic evolution of ligands by exponential enrichment with integrated optimization by non-linear analysis. Journal of Molecular Biology **222**(3), 739–761 (1991).
38. Maung, N.W., Christina, D.S.: Higher-order cellular information processing with synthetic RNA devices. Science **322**(5900), 456–460 (2008).
39. Khalid, K.A., Kwaku, D.T., Matthew, F.L., et al.: A fluorescent split aptamer for visualizing RNA-RNA assembly *in vivo*. ACS Synthetic Biology **6**(9), 1710–1721 (2017).
40. Deltcheva, E., Chylinski, K., Sharma, C.M., et al.: CRISPR RNA maturation by trans-encoded small RNA and host factor RNase III. Nature **471**(7340), 602–607 (2011).
41. Breanna, D.A., Noreen, W., Eileen, H., et al.: PERSIST: A programmable RNA regulation platform using CRISPR endoRNases. bioRxiv, 12.15.867150 (2019).
42. Kawasaki, S., Ono, H., Hirosawa, M., et al.: Programmable mammalian translational modulators by CRISPR-associated proteins. Nature Communications **14**(1), 2243 (2023).
43. Chen, S., Gong, B., Zhu, C., et al.: Nucleic acid-assisted CRISPR-Cas systems for advanced biosensing and bioimaging. TrAC Trends in Analytical Chemistry **159**, 116931 (2023).
44. Gardner, T., Cantor, C., Collins, J.: Construction of a genetic toggle switch in Escherichia coli. Nature **403**, 339–342 (2000).
45. Elowitz, M.B., Liebler, S.: A synthetic oscillatory network of transcriptional regulators. Nature **403**(1), 335–338 (2000).
46. Ausländer, S., Ausländer, D., Müller, M., et al.: Programmable single-cell mammalian biocomputers. Nature **487**(7405), 123–127 (2012).
47. Wroblewska, L., Kitada, T., Endo, K., et al.: Mammalian synthetic circuits with RNA binding proteins delivered by RNA. Nature Biotechnology **33**, 839–841 (2015).

48. Green, A.A., Kim, J., Ma, D., et al.: Complex cellular logic computation using ribocomputing devices. Nature **548**, 117–121 (2017).
49. Matsuura, S., Ono, H., Kawasaki, S., et al.: Synthetic RNA-based logic computation in mammalian cells. Nature Communications **9**, 4847 (2018).

Chapter 12
Protein Computing

Feynman's vision of "developing computers at the molecular scale" led to the birth of the DNA computing model in 1994, followed by protein computing in 1995: a protein computing model of 2-state logic gates was proposed. Since then, many scholars have studied numerous protein logic gates, logic calculators, arithmetic calculators, protein computing models for solving **NP**-complete problems, protein storage and computing devices, etc. This chapter introduces some typical representatives.

12.1 Introduction

Proteins have a myriad of structures and can perform a variety of functions in the body, making them natural high-performance biomaterials. Any protein that can transform input signals into output signals can be considered an element that performs some computation or carries some information [1]. Since the early 1990s, people have begun to explore the use of proteins to store information, and in recent years, the exploration of using different proteins to build memristors. Nicolau et al. explored the solution of NP-complete problems by protein computing in 2016 [2], using protein molecules (actin and tubulin) to solve a specific mathematical problem (subset sum problem). Most of the work on protein computing focuses on exploring the construction of logic calculators and arithmetic calculators with proteins. This chapter will introduce in detail from four aspects: building logic calculators based on proteins, building arithmetic calculators based on proteins, solving **NP**-complete problems based on protein molecules, and protein storage.

J. Xu, *Biological Computing*, https://doi.org/10.1007/978-981-96-3870-3_12

12.2 Building Logic Calculators Based on Proteins

Logic gates are a fundamental component of computers. In electronic Boolean logic gates, low voltage/current corresponds to 0, and high voltage/current corresponds to 1. This concept can be extended to biochemical reactions, where low concentration/activity is 0, and high concentration/activity is 1. Although the inputs and outputs of molecular logic gates are not truly binary, molecular circuits can be designed to approximate binary logic gates, responding to inputs in an S-shaped response [3]. Protein-based logic calculators are usually constructed using enzyme-mediated reactions [3, 4], protein interactions [5, 6], protein conformational effects [7, 8], post-translational modifications of proteins [9], and proteins designed and modified artificially [5, 6, 9, 10], among which the most studied are enzyme-mediated logic calculators. These are introduced below.

12.2.1 Enzyme-Mediated Logic Calculators

In the process of constructing a protein logic calculation system, the design and implementation of enzyme-based logic gates have become a focus of attention. Firstly, enzymes exhibit excellent specificity and selectivity, making them more reliable and accurate in biological computing tasks due to their recognition of specific substrates. Secondly, as biological catalysts, enzymes can efficiently catalyze biochemical reactions, executing specific logic operations more quickly. Moreover, enzymes are naturally occurring molecules in the body, unlikely to cause immune reactions, which helps to realize biological computing in the body. In addition, the activity of enzymes can be precisely controlled by adjusting factors such as pH and temperature, providing convenience for the design of adjustable biological computing systems. Enzymes also have the ability to self-assemble, interacting with other molecules or nanomaterials, providing the possibility to construct more complex biological computing devices [11]. Enzyme-based logic gates usually use relatively simple enzyme-catalyzed reactions for implementation, which are more intuitive in design. The rapid development of enzyme-based information processing systems has promoted the design of various Boolean logic gates, including AND, OR, NAND, NOR, CNOT, XOR, INHIBIT, YES, NOT, etc. Researchers have also designed various cascade reactions to simulate combinations of different logic gates [12].

12.2.1.1 General Definition of Enzyme-Based Logic Calculators

Biochemical reactions are not binary processes in their essence, so to simulate binary computing, especially the implementation of Boolean logic gates, special processing methods need to be adopted to perform binary operations. Low and

high concentrations of chemicals can be correspondingly defined as 0 and 1. When there are no reactants in the system, it is usually considered as the logical 0 of the input signal. In experiments, the high concentration corresponding to logic 1 can vary depending on the type of reaction and the method used to analyze the resulting chemical changes. In some specific application scenarios, such as in the use of biomedical/biosensors, the low concentration of initial reagents corresponding to logic values 0 and 1 can be set to high concentration for natural reasons. For example, logic 0 and 1 inputs can be defined as the normal physiological concentration and abnormal pathological concentration of reactant species. In this case, the gap between logical inputs 0 and 1 may be small, resulting in relatively small differences in output signals, increasing the complexity of binary discrimination.

In addition to the substrate concentration of the enzyme as input 0 or 1 [13], the presence or absence of the enzyme can also serve as input 0 or 1 [12, 14]. When the substrate is used as input, the entire reaction system can use binary variable low molecular weight substances (substrates, cofactors), biological catalyst species (enzymes), and some auxiliary reagents (starters, salts, buffers, etc.) as the "mechanical" part of the logic gate. The definition of logical input and mechanical part can change, and enzymes can also serve as logically variable inputs, while other reactants are considered as non-variable "mechanical" parts.

12.2.1.2 Enzyme-Based Boolean Logic Gates

By choosing different combinations of enzymes and reaction types, different logic gates can be realized. Simple Boolean logic gates process one or two inputs to produce one output. Common simple logic gates include: YES, NOT, OR, NOR, XOR, NXOR, AND, NAND, INHIBIT. Different enzymes can simulate the same logic operation, and the following introduces the above 9 simple logic gates implemented by enzyme-catalyzed reactions.

YES Gate

The YES gate is the simplest logic gate, with a single input and output. When the input is logic 0, the output is logic 0; when the input is logic 1, the output is logic 1. The biochemical reaction implementation is also the simplest, any chemical reaction that produces a chemical product in the presence of the corresponding original substrate can be considered a YES gate. Many different enzyme-catalyzed reactions have been reported to simulate this simple logic operation [15]. As shown in Fig. 12.1, in the presence of lactate (Lac), lactate dehydrogenase (LDH) catalyzes the reduction of NAD^+ to NADH, which is a simple YES gate. NADH has the best absorbance at $\lambda = 340$ nm, and NAD^+ has weak absorbance at $\lambda = 340$ nm, so by detecting whether the absorbance at $\lambda = 340$ nm increases, it can be determined whether NADH is produced [13, 16]. When Lac is present (input is 1), NADH

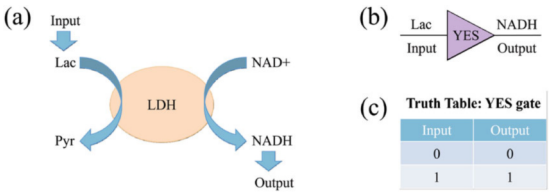


Fig. 12.1 Schematic diagram of the YES gate. (**a**) The YES gate based on enzyme catalytic reactions. (**b**) The scheme of the YES gate. (**c**) Truth table of Boolean YES gate

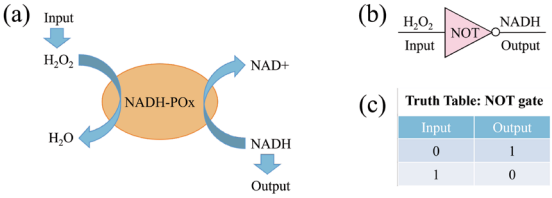
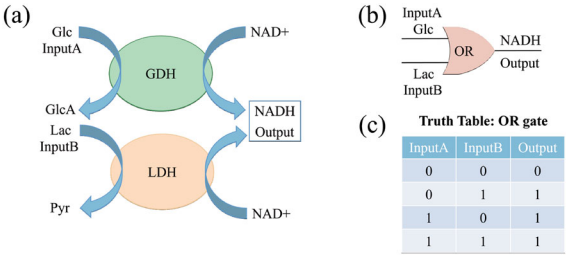


Fig. 12.2 Schematic diagram of the NOT gate. (**a**) The NOT gate based on enzyme catalytic reactions. (**b**) The scheme of the NOT gate. (**c**) Truth table of Boolean NOT gate

is produced, causing an increase in absorbance at $\lambda = 340$ nm. When the increase in absorbance exceeds a certain threshold, it is defined as output signal 1. In the absence of Lac (input is 0), the reaction cannot proceed, the original NAD^+ has a small absorbance at $\lambda = 340$ nm and does not change, which is defined as output signal 0. This YES gate simulation process uses a standard biological analysis method and is simple to operate. Although the YES gate is relatively simple, it is used in complex logic systems (such as the logically reversible Feynman, Double Feynman, Toffoli, Peres, and Fredkin gates play an important role in them [12], and they are also indispensable components in biological computing systems.

NOT Gate

The NOT gate is similar to the YES gate, both are single-input and single-output, but have a reverse output signal. When the input is logic 0, the output is logic 1; when the input is logic 1, the output is logic 0. The NOT gate can be realized through various enzyme-catalyzed reactions [12, 13]. As shown in Fig. 12.2, in the presence of hydrogen peroxide (H_2O_2), Nicotinamide Adenine Dinucleotide Phosphate Oxidase 2 (NADH-POx) oxidizes NADH to NAD^+. In the presence of H_2O_2 (input is 1), NADH decreases, and the absorbance at $\lambda = 340$ nm decreases, defining this absorbance change as 0; in the absence of H_2O_2 (input is 0), the reaction does not occur, NADH remains unchanged, and the absorbance at $\lambda = 340$ nm remains unchanged, defining it as output 1, this process is similar to the logical function NOT. Although the NOT gate is simple, it is a very important part of various biological computing systems, for example, it operates as an inverter in a half subtractor [11, 13].



InputA	InputB	Output
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 12.3 Schematic diagram of the OR gate. (**a**) The OR gate based on enzyme catalytic reactions. (**b**) The scheme of the OR gate. (**c**) Truth table of Boolean OR gate

OR Gate

The OR gate is a logic gate with dual inputs and single output. When at least one input is 1, the output is 1; when both inputs are 0, the output is 0. Its design implementation is relatively easy, and can be realized by two biological catalytic reactions that generate the same chemical product. Figure 12.3 shows two biological catalytic reactions that generate NADH, one catalytic reaction produces NADH through Glucose Dehydrogenase (GDH), and the other produces NADH through Lactate Dehydrogenase (LDH), by detecting the change in absorbance at $\lambda = 340$ nm to detect the output signal. When Glc (input A) and Lac (input B) at least one reducing substrate is present (input is: 1, 0; 0, 1; 1, 1), NADH can be generated, causing an increase in absorbance (output is 1). When Glc and Lac are both absent (input is 0, 0), the reaction does not occur, the concentration of NADH remains unchanged, and the absorbance remains unchanged (output is 0). Because the implementation of the OR gate is simple, the enzyme-based OR logic gate is one of the most frequently reported logic systems, with a wide range of applications.

NOR Gate

The NOR gate is basically the same as the OR gate, with dual inputs and single output, but has a reverse output signal. When both inputs are 0, the output is 1; when at least one input is 1, the output is 0. This logic gate can be constructed when both input reagents inhibit the same biological catalytic reaction. Although the design of the NOR gate is feasible, it is relatively difficult to implement. If the generation of a substance is defined as output 1 like the OR gate, in addition to the two normal catalytic reactions, two additional inhibitors (inputs) need to be added to realize the NOR gate, and the reaction is very complex. Therefore, defining the consumption of a substance as output 0, the implementation of the NOR gate will be simpler. Figure 12.4 shows two biological catalytic reactions that consume NADPH. NADPH is similar to NADH, with the best absorbance at $\lambda = 340$ nm, and $NADP^+$ has weak absorbance at $\lambda = 340$ nm. The reaction catalyzed by Glutathione Reductase (GR) starts in the presence of oxidized glutathione (GSSG) (input A is 1), causing NADPH to oxidize, the concentration of NADPH decreases, and the absorbance at $\lambda = 340$ nm decreases (output is 0). The second reaction is catalyzed

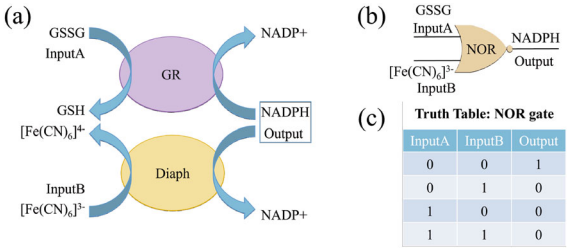


Fig. 12.4 Schematic diagram of the NOR gate. (**a**) The NOR gate based on enzyme catalytic reactions. (**b**) The scheme of the NOR gate. (**c**) Truth table of Boolean NOR gate

by Diaphorase (Diaph), in the presence of $[Fe(CN)_6]^{3-}$ (input B is 1), the same output result is obtained. Therefore, when at least one input reagent is present (input is: 0, 1; 1, 0; 1, 1), the absorbance decreases (output is 0); and when both input reagents are absent (input is: 0, 0), the absorbance remains at the original high value (output is 1). The current design of the NOR gate makes it easy to integrate into complex logic networks, such as multiplexers, and applications in biological sensor alarm devices. In addition, Chuang pointed out that the NOR logic gate is a universal gate, which can be used to construct logic circuits that perform all other logic operations [17].

XOR Gate

The XOR gate is one of the key elements in designing complex logic systems (such as reversible logic gates and arithmetic operations). When the two inputs are inconsistent (inputs are 1, 0 or 0, 1), the XOR gate outputs 1; when the two inputs are consistent (inputs are 0, 0 or 1, 1), the XOR gate outputs 0. The XOR gate has been implemented using many different enzymes [13, 18–20], but implementing it in a biocatalytic system presents certain challenges, usually requiring the introduction of an "artificial" assumption—that is, the output signal usually needs to use the absolute value of the generated signal. By carrying out both product multiplication and product reduction biocatalytic reactions, the implementation of the XOR gate becomes possible. Figure 12.5 shows an example of an XOR gate at work, using the absolute value of the change in absorbance as the output signal. NADH-POx causes NADH oxidation in the presence of H_2O_2 (input A), reducing NADH and lowering the absorbance at $\lambda = 340$ nm. LDH causes NAD^+ reduction in the presence of Lac (input B), generating NADH and increasing the absorbance at $\lambda = 340$ nm. In the absence of both H_2O_2 and Lac (inputs are 0, 0), the light absorbance remains unchanged (output 0). The presence of either H_2O_2 or Lac (inputs are: 1, 0; 0, 1) will activate one of the reactions, respectively reducing or increasing the concentration of NADH and its absorbance. Although the changes in concentration and absorbance are different (opposite), if the absolute value of the change in absorbance is defined as the output signal, these two results can be considered as the same output 1. For XOR logic operations, it is crucial to produce a low response to the two inputs (inputs are 1, 1) (output 0). It is necessary to optimize the concentrations

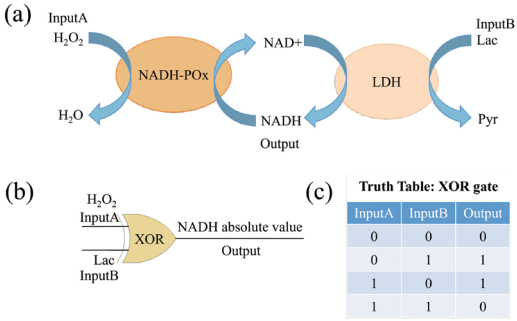
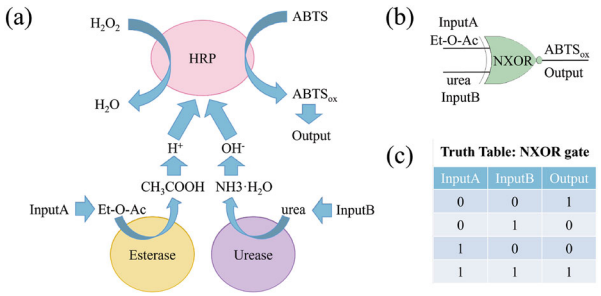


Fig. 12.5 Schematic diagram of the XOR gate. (**a**) The XOR gate based on enzyme catalytic reactions. (**b**) The scheme of the XOR gate. (**c**) Truth table of Boolean XOR gate

of the two inputs to ensure that the rates of NADH consumption and generation are consistent, thus keeping the concentration of NADH and its absorbance almost unchanged. The method of using "artificial" definition of absolute value change as the output signal has advantages and disadvantages. The advantage is that it makes the implementation of the XOR gate relatively simple, the disadvantage is that the XOR gate has some limitations in connecting logic gate networks. The absolute value change can be used as the final signal produced by logic gates connected by cascading reactions, but because two different chemicals (such as NAD^+ and NADH) cannot participate in a reaction in the same way, this gate cannot be extended to other logic operations.

NXOR Gate

The NXOR gate is similar to the XOR gate, but has a reverse output signal [21]. When the two inputs are inconsistent (inputs are 1, 0 or 1, 0), the NXOR gate outputs 0; when the two inputs are consistent (inputs are 0, 0 or 1, 1), the NXOR gate outputs 1. However, individual inputs (inputs are 0, 1 and 1, 0) inhibit the biocatalytic process, resulting in a low output signal of logical value 0. Figure 12.6 provides a possible method for implementing this process. Horseradish peroxidase (HRP) catalyzes the oxidation of the substrate 2,2'-azino-bis(3-ethylbenzothiazoline-6-sulfonic acid) (ABTS), generating the oxidized state of ABTS (ABTSox), which has the best absorbance at $\lambda = 420$ nm, so the change in absorbance at $\lambda = 420$ nm is used as the output signal. This reaction is continuous, and HRP maintains optimal activity at a specific pH value. The esterase-catalyzed hydrolysis of Et-O-Ac (input A) to produce CH_3COOH and the urease-catalyzed hydrolysis of urea (input B) to produce $NH_3 \cdot H_2O$ can regulate the pH of the solution. When both Et-O-Ac and urea are absent (inputs are 0, 0), or when the amounts of Et-O-Ac and urea are comparable (inputs are 1, 1), the produced CH_3COOH and $NH_3 \cdot H_2O$ neutralize each other, and the pH of the solution does not fluctuate significantly, maintaining the optimal activity of HRP, and ABTSox continues to be produced, increasing the absorbance at $\lambda = 420$ nm (output is 1). However, when the concentrations of Et-O-Ac and urea differ significantly (inputs are 1, 0 or 0, 1), the pH changes, HRP



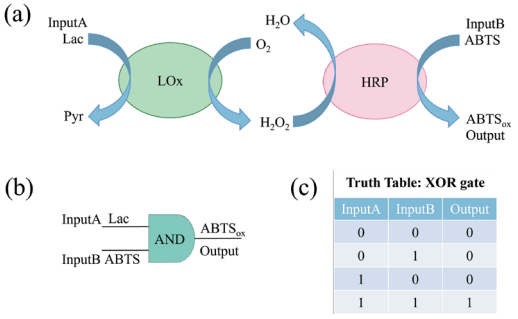
InputA	InputB	Output
0	0	1
0	1	0
1	0	0
1	1	1

Fig. 12.6 Schematic diagram of the NXOR gate. (**a**) The NXOR gate based on enzyme catalytic reactions. (**b**) The scheme of the NXOR gate. (**c**) Truth table of Boolean NXOR gate

activity decreases, ABTSox production decreases, and the absorbance at $\lambda = 420$ nm is low (output is 0). Enzymes whose activity depends on pH can be used to design similar NOXR gates, and the optimal activity pH is best near neutral pH, so that the pH can move in both directions.

AND Gate

The AND gate is one of the most commonly designed logic gates, especially those implemented through enzymatic reactions. When both inputs are 1 (inputs are 1, 1), the AND gate outputs 1; in other cases (inputs are 0, 1; 1, 0; 0, 0), the AND gate outputs 0. AND gates are usually designed as a cascade of two consecutive biocatalytic processes, and Fig. 12.7 is an AND gate composed of a cascade reaction. Lac (input A) is catalyzed by lactic acid oxidase (LOx) to produce H_2O_2. H_2O_2 oxidizes ABTS (input B) to produce ABTSox, which serves as the final output signal. Only when both inputs Lac and ABTS are present (inputs are 1, 1), the cascade reaction can be completed at once, producing ABTSox and increasing the absorbance at $\lambda = 420$ nm (output is 1). The number of biocatalytic reactions and the number of enzymes involved may vary, all of which can be used to construct AND



InputA	InputB	Output
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 12.7 Schematic diagram of the AND gate. (**a**) The AND gate based on enzyme catalytic reactions. (**b**) The scheme of the AND gate. (**c**) Truth table of Boolean AND gate

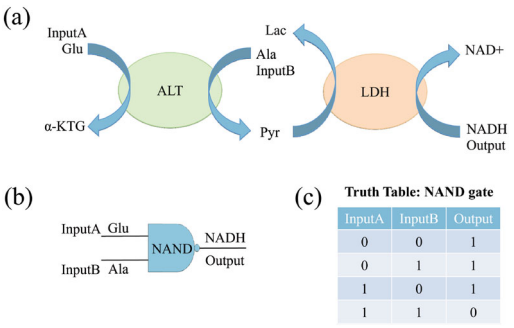


Fig. 12.8 Schematic diagram of the NAND gate. (**a**) The NAND gate based on enzyme catalytic reactions. (**b**) The scheme of the NAND gate. (**c**) Truth table of Boolean NAND gate

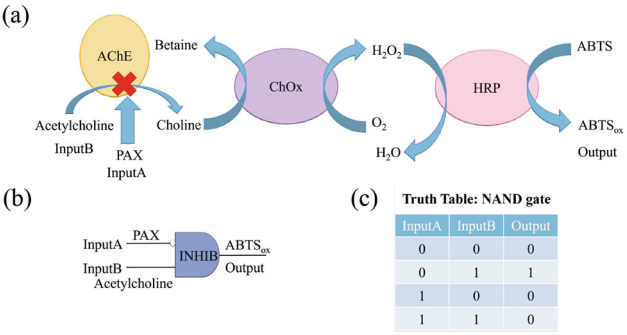
gates. In the simplest implementation of AND logic, the two substrates of a single enzyme represent two input signals [14]. Enzyme-based logic AND gates have been widely used in various biosensing systems.

NAND Gate

The NAND gate is similar to the AND gate, but the output signal is reversed. When both inputs are 1 (input is 1,1), the AND gate outputs 0; in other cases (input is 0,1; 1,0; 0,0), the AND gate outputs 1. In specific biological catalytic reactions, reversal can be achieved through logical operations [22]. The simplest method is to change the definition of the output signal, taking the consumption of chemical molecules (concentration decrease) as output signal 1. Figure 12.8 shows the cascade process of two enzymes catalyzed by two different input signals. Under the catalysis of glutamate transaminase (ALT), glutamate (Glue, input A) and alanine (Ala, input B) are respectively transformed into α-ketoglutarate (α-KTG) and pyruvate (Pyr). This reaction only starts when both Glu and Ala inputs are present (input is 1,1), similar to the AND logic gate. When Pyr is generated in the first step of the reaction, it is reduced to Lac in the LDH catalytic process, causing NADH to be oxidized to NAD^+, and the absorbance at $\lambda = 340$ nm decreases (output is 0). When the input reactants are missing (input is 0,1; 1,0; 0,0), the intermediate product Pyr cannot be produced, the subsequent reaction cannot start, the amount of NADH does not change, and the absorbance does not decrease (output is 1).

INHIB Gate

The INHIB gate is a special logic gate, where one input can inhibit the logic operation when it is 1. As shown in Fig. 12.9, when input A is 0, the output of the INHIB gate is determined by input B, which is equivalent to a YES gate based on input B; when input A is 1, no matter what input B is, the output of the INHIB gate is 0, that is, when input A is 1, the logic operation of the INHIB gate is inhibited and can only output 0. In the example in Fig. 12.9, acetylcholinesterase catalyzes the hydrolysis of acetylcholine (input B), generating choline and acetic



InputA	InputB	Output
0	0	0
0	1	1
1	0	0
1	1	0

Fig. 12.9 Schematic diagram of the INHIB gate. (**a**) The INHIB gate based on enzyme catalytic reactions. (**b**) The scheme of the INHIB gate. (**c**) Truth table of Boolean INHIB gate

acid. Subsequently, choline is oxidized in the reaction catalyzed by choline oxidase, simultaneously producing H_2O_2. H_2O_2 oxidizes ABTS in the reaction catalyzed by horseradish peroxidase (HRP), generating ABTSox as the output signal. However, pralidoxime (PAX) (input B) inhibits the hydrolysis of acetylcholine, preventing the subsequent reaction from continuing. When PAX is absent (input A is 0), and there is no acetylcholine (input B is 0), the entire biocatalytic cascade reaction is in a silent state, unable to produce ABTSox (output is 0); when acetylcholine is present (input B is 1), the reaction occurs in succession, generating ABTSox (output is 1). When PAX is present (input A is 1), it inhibits this series of reactions, and does not generate ABTSox (output is 0). The INHIB logic gate built on the inhibitory effect of PAX can be implemented in the biocatalytic cascade, without directly inhibiting the enzyme, but still performing the same logic function [13].

12.2.1.3 Enzyme-Based Logic Circuits

Enzyme-based logic gates are realized through enzymatic catalysis. By designing specific interactions between enzymes and substrates, the cascading of enzyme-based logic gates can be achieved, forming complex logic circuits. As shown in Fig. 12.10, Tamara et al. assembled OR, AND, XOR three logic gates by combining acetylcholinesterase (AChE), choline oxidase (ChOx), microperoxidase-11 (MP-11), glucose dehydrogenase (GDH) four enzymes and acetylcholine, butyrylcholine, oxygen, glucose four substrates, and cascaded into a logic circuit, with the change in absorbance of NADH at $\lambda = 340$ nm as the output signal [23]. Acetylcholine (input A) or butyrylcholine (input B) is hydrolyzed under the catalysis of AChE, and betaine aldehyde is reduced to choline, forming an OR gate. Choline and oxygen (input C) generate betaine aldehyde and H_2O_2 under the catalysis of ChOx, forming an AND gate. When H_2O_2 is present, NADH is oxidized to NAD^+ under the catalysis of MP-11, NADH decreases, and the absorbance decreases (output is 1); on the other hand, when glucose (input D) is present, NAD^+ is reduced to NADH under the catalysis of GDH, NADH increases, and the absorbance increases

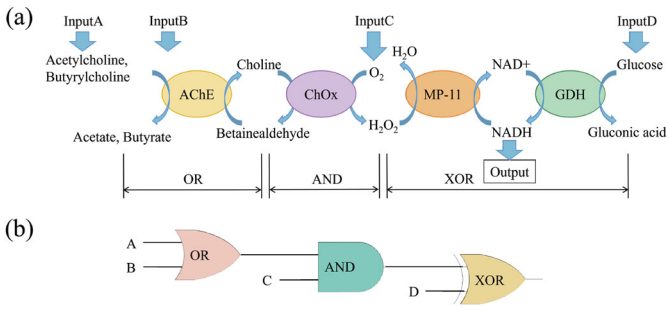


Fig. 12.10 Logic circuit composed of four enzymes in series. (**a**) The Logic circuit based on enzyme catalytic reactions. (**b**) The scheme of the Logic circuit

(output is 1); when H_2O_2 and glucose are either both absent or both present, the amount of NADH does not change, and the output is 0; forming an XOR gate. The output of one logic gate serves as the input of the next logic gate, realizing the cascading of logic gates and constructing complex logic circuits. To ensure that the logic circuit reacts according to the design, the concentration of the enzyme and substrate needs to be balanced. To achieve this, many optimization experiments have been conducted on the concentration of enzymes and input substrates.

However, cascading enzyme-based logic gates may have the following problems:

Signal transmission delay: Each logic gate's catalytic reaction requires a certain amount of time to complete. When multiple logic gates are cascaded, the output of each gate will become the input of the next gate, leading to signal transmission delay. This may be unacceptable in some applications, especially for systems that require a quick response. Signal Attenuation: Long-term cascading may lead to signal attenuation. Since each logic gate's response introduces some noise and loss, these effects may accumulate during transmission, affecting the system's stability and accuracy.

Mutual Influence There may be mutual influence between cascaded logic gates. The output substance may affect the catalytic reaction of the next logic gate, thereby changing the behavior of the entire system. This requires careful design and optimization to ensure that the output of each logic gate does not adversely affect the input of the next logic gate.

Substrate Depletion When cascading multiple logic gates, substrate depletion may become a problem. If the substrate of a logic gate is the reactant of the next logic gate, then the decrease in substrate concentration may affect the performance of the entire system.

Despite these issues, researchers have successfully implemented cascaded enzyme-based logic gates through careful design and adjustment of reaction conditions. These systems may play an important role in the fields of biological computation and biosensors, but they need to be carefully designed and optimized in specific applications.

12.2.2 Non-enzyme Mediated Logic Operators

Non-enzyme mediated logic operators can be divided into types such as receptor-ligand interactions, conformational effects, and post-translational modifications, depending on the mechanism and process by which proteins perform functions [1].

12.2.2.1 Receptor-Ligand Interactions

Receptor-ligand interactions are an important type of molecular interaction in biology, usually involving specific binding between proteins (receptors) and ligands. This interaction can trigger intracellular cascade reactions, thereby affecting the physiological functions and behaviors of cells. It plays a key role in biological processes such as cell signal transduction, immune response, hormone regulation, and is crucial for the perception and response to the internal and external environment of cells.

Similar to enzyme-based protein logic computation, receptor-ligand interactions can also be viewed as a signal processing process, where receptor proteins read the concentration or specific features of ligands and generate corresponding intracellular signals. This signal processing process is similar to the input-output relationship in computation, so the mechanism of receptor-ligand interactions can also be used to design logic gates.

Ronde and others defined ligand concentration as input and receptor protein activity as output, realizing 16 types of logic gates [24]. Receptors usually exist in the form of dimers or higher polymers, and most logic gates can be realized by two single-ligand receptors or one double-ligand receptor. Smita and others designed an antibody logic gate that combines phosphorylation. The 104th residue of the complementary region (CDRs) of the single-domain antibody cAb-Lys3 is the key residue for binding to the antigen lysozyme. Phosphorylating the 104th residue can only be dephosphorylated in the presence of a phosphatase, binding to the antigen, realizing an AND logic gate, which can control the immune response [25]. Simone and others used antigen-antibody interactions, targeting the CD52 and CD20 cell surface proteins of tumor B cells, introduced mutations G236R and G237A on the antibody IgG, and when CD52 and CD20 were recognized at the same time, hetero-oligomers would form, enhancing the affinity with protein C1q and FcγR, executing effector functions: cell lysis and inflammatory response, cytotoxicity and cell phagocytosis, etc., also realizing an AND logic gate [26].

Processes similar to receptor-ligand interactions also include protein-protein interactions. Tae believes that the transcription process is an AND gate. When the transcription factor and its partner protein are present at the same time (input signal: 1,1), a protein can be expressed (output is 1). This protein can be the input of the next AND gate, so it can be cascaded into a multi-input AND gate [27]. Y406A is a pore-forming protein that forms pores on the lipid membrane under low pH conditions. The ankyrin repeat domain protein inhibitor (D22) also inhibits

the function of Y406A when it binds to Y406A. D22 reversibly binds to Y406A. D22 is cut on the membrane (dissociated from the membrane by TCEP or MMP-9 cutting on different lipid membrane systems), dissociates from Y406A, and Y406A can form pores under the condition of simultaneously satisfying low pH. Based on this principle, Omersa and others designed AND gates and OR gates [28].

12.2.2.2 Conformational Effects

Conformational effects refer to the process where the binding of a ligand to a site on a protein changes the protein's conformation, leading to a change in the protein's biological activity, thereby achieving corresponding functional regulation [7]. Conformational effects play an important role in many biological processes such as signal transduction, transcription regulation, and metabolism. The conformational change of a protein can be regulated by external stimuli or internal signals. This change can be seen as a computational process, that is, the input signal (stimulus or signal) is transformed into a specific output response (conformational change). Conformational effects can be used to implement logic operations, signal transmission, and information processing, and thus can be applied to protein computation [8].

Nikolay and others have found that nanocomputing devices (NCA) built with single-chain proteins use individual proteins or protein structural domains as response units (RU); inputs can be provided by many functional modulators, such as light, drugs, PH, temperature, RNA sensitive functional modulators; protein conformation changes or active site changes (binding ability changes) are used as outputs [8]. In recent years, there have been many works using light-sensitive structural domains (LOV) and drug-sensitive structural domains (uniRapR) to regulate the binding activity of response units [6, 8–10, 29]. As shown in Fig. 12.11, the light-sensitive modulator (LFM) receives light irradiation, causing disorderly fluctuations in the active site of RU, inhibiting the binding activity of RU and substrate (A); when the drug-sensitive modulator (DFM) binds with the drug molecule rapamycin, it changes from disorder to order, promoting the binding of RU and substrate (B); inserting a self-inhibitory structural domain (AID) on RU, AID occupies the binding site of RU substrate, causing RU to not bind with the substrate. LFM receives light irradiation, which will cause AID to dissociate, allowing RU to successfully bind with the substrate (C). RU is split into N and C ends, each connected to a part of the dimeric protein (iFKBP and FRB), in the presence of rapamycin, iFKBP and FRB form a dimeric protein, and the N and C ends of RU also bind together, forming a functional RU that can bind to the substrate (D). Through the combination and modification of light-sensitive structural domains and drug-sensitive structural domains, various logic gates, such as AND, OR, etc., can be constructed to regulate protein conformation, thereby directly controlling functions.

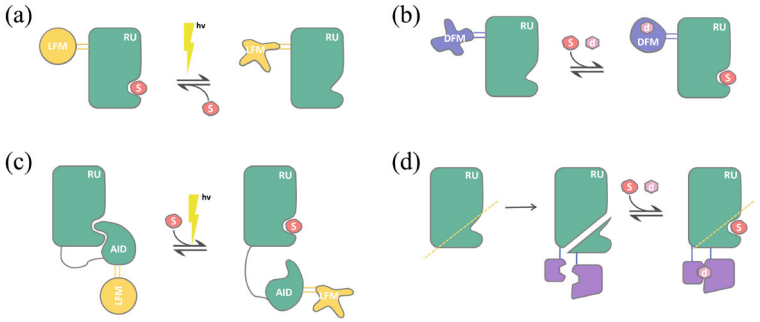


Fig. 12.11 Some of the established modes of protein control. (**a**) Photo-allosteric inhibition. (**b**) Chemo-allosteric inhibition. (**c**) Photo-allosteric activation. (d) Controlling protein function via split reassembly. This figure originally published by Dokholyan NV. Nanoscale programming of cellular and physiological phenotypes: inorganic meets organic programming. NPJ Syst Biol Appl. 2021 Mar 11;7(1):15. https://doi.org/10.1038/s41540-021-00176-8; released under a [CC licence type, eg. Creative Commons Attribution 4.0 International License (CC BY 4.0)]

12.2.2.3 Post-translational Modification

Post-translational modification refers to the modification of proteins after synthesis through chemical methods, including phosphorylation, methylation, nucleotidylation, acylation, etc. Post-translational modification can change the structure, function or interaction of proteins, thereby affecting intracellular signal transduction, metabolic regulation and cell function [1]. This regulatory process can also be used to construct logic gates. For example, the kinase DRP-1 is active only when it forms a dimer, and can phosphorylate other monomers, and both monomers are deactivated when phosphorylated [30]. Unger and others connected the kinase DRP-1 monomer to a specific DNA chain, including two input DNA tags, one output DNA tag, and initially the output tag was blocked by another complementary DNA chain. This DRP-1 monomer constitutes a NAND gate, relying on the input DNA tags on the gate, two monomers approach and form a dimer, phosphorylating the gate (output is 1); but when both monomers are in a phosphorylated state (input combination: 1, 1), the gate cannot be phosphorylated (output is 0). The formation of two monomers into a dimer can activate the nuclease structural domain, hydrolyze the blocking DNA chain, and allow the output gate to continue to participate in the next level of reaction. Through the design combination of different DNA tags, different scales of biomolecular logic circuits can be realized [30]. Smita and others used the phosphate group to occupy the complementary determining region (CDR) of the antibody to achieve antigen-antibody binding, and finally realized the AND gate [25]. Post-translational modification regulates the function or interaction of proteins by regulating the activity or structure of proteins, and can construct different logic systems, which have broad application prospects in the fields of biosensing, drug delivery and treatment, molecular computing, etc.

12.2.3 Logic Calculators Based on Artificially Designed Proteins

Although there are many types of natural proteins, which can perform diversified tasks in protein computation, their functions and structures are aimed at specific biological environments and biological processes, and cannot flexibly adapt to the computational needs of artificial design, so in some cases, engineering modifications and de novo design of proteins are needed to achieve new computational functions. A common method is to modify the specific sites of proteins, such as inserting specific sites on DNA according to computational needs, these sites can bind with repressors, transcription factors and cofactors etc. to achieve the purpose of regulating target gene expression [3, 31, 32]. In recent years, with the rapid development of synthetic biology and protein engineering technology, research on protein computation is no longer limited to the modification of natural proteins, but can also design new proteins with specific structures and functions from scratch. David The Baker research group has successfully combined various homodimeric proteins, designed from scratch, into basic units of logic gates [5]. Different protein monomers are combined into logic gates, and the inputs are corresponding heterodimers, connected by linkers. This design makes it easier for the input protein monomers to form homodimers with the protein monomers on the gate. In this system, the input heterodimers can be considered as the input signals of the logic gate. By combining the input signals, new homodimeric proteins can be formed to construct different logic gates. The output can be reflected by measuring the size and concentration of the new protein. Specifically, when the optical density (OD value) of the protein is large enough, it can be considered as the output signal of the logic gate is 1. This design not only provides a novel protein calculation method, but also provides a feasible framework for the construction of biological logic gates. Such protein modification and design methods have expanded our understanding of protein engineering and provided more possibilities for customized biological systems and molecular computing.

12.3 Building Arithmetic Calculators Based on Proteins

The cascading of simple protein-based logic gates can not only construct more complex logic gates, such as Feynman, Double Feynman, Toffoli, Peres, Fredkin and other reversible logic gates [14], but also can form half adders and half subtractors [13]. Half adders and half subtractors are basic components in digital circuits, used to perform addition or subtraction operations of two single-bit binary numbers. A half adder requires an XOR gate and an AND gate in cascade, producing two output results, namely the sum bit (Sum) and the carry bit (Carry); while a half subtractor requires an XOR gate and an INHIB gate in cascade, producing the difference bit (Difference) and the borrow bit (Borrow). Usually, enzymes have

high selectivity for specific substrates, but sometimes unexpected substrate cross-reactions occur, which may lead to unexpected products in the design of complex enzyme cascade systems, thereby affecting the accuracy and controllability of the computing system [11].

To solve the above problem, Brain and others used microfluidic chips to fix each enzyme in their own flow pools separately, and then connected different flow pools to form different logic gates, forming different output bits of half adders and half subtractors [11]. The half adder is composed of two logic gates, but these two logic gates are built separately, with the same input but no mutual influence. Diaphorase (Diaph) reduces $[Fe(CN)_6]^{3-}$ to $[Fe(CN)_6]^{4-}$ in the presence of NADH (input A is 1); Glucose oxidase (GOx) reduces O_2 to H_2O_2 in the presence of glucose (Glc) (input B is 1); H_2O_2 enters the next flow pool, and under the catalysis of horseradish peroxidase (HRP), $[Fe(CN)_6]^{4-}$ is oxidized to $[Fe(CN)_6]^{3-}$. The two reactions in cascade form an XOR gate. $[Fe(CN)_6]^{3-}$ has the strongest absorbance at $\lambda = 420$ nm. Only when NADH is input (input signal: 1, 0), $[Fe(CN)_6]^{3-}$ decreases, and the absorbance decreases (output is 1); only when Glc is input (input signal: 0, 1), $[Fe(CN)_6]^{3-}$ increases, and the absorbance increases (output is 1); when neither or both are input (input signal: 0, 0; 1, 1), the concentration of $[Fe(CN)_6]^{3-}$ remains unchanged, and the absorbance remains unchanged (output is 0). Lactate dehydrogenase (LDH) oxidizes NADH to NAD^+ in the presence of NADH (input A is 1); NAD^+ enters the next reaction pool, and glutamate dehydrogenase (GDH) reduces NAD^+ to NADH in the presence of Glc (input B is 1), and NADH enters the next reaction pool, Diaphorase (Diaph) reduces $[Fe(CN)_6]^{3-}$ to $[Fe(CN)_6]^{4-}$ in the presence of NADH. The three enzyme reactions in series form a complex AND gate. Only when both NADH and Glc are present (input signal: 1, 1), can the reaction in the last reaction pool occur, $[Fe(CN)_6]^{3-}$ decreases, and the absorbance decreases (output is 1). The XOR gate and AND gate have consistent input signals, and the reactions are separate, giving the sum bit and carry bit values separately, realizing the half adder. The implementation of the half subtractor is similar to the half adder, by cascading reaction pools, separately constructing XOR gate (difference bit) and INHIB gate (borrow bit), to realize the half subtractor.

Multiple half adders or half subtractors can be combined into a full adder or full subtractor to perform more complex arithmetic operations. However, enzymes can also be used directly for addition, subtraction, and multiplication operations [33]. Ivanov and others fixed enzymes on hydrogel beads and separated them with a continuous stirred tank reactor (CSTR). According to different enzymes, polypeptide chains are formulated as inputs. The polypeptide chains have specially made cleavage sites that can be cut by specific enzymes. The other end of the cleavage site is connected with a fluorescent substance (AMC). The polypeptide chain enters the CSTR and reacts under the catalysis of the enzyme. AMC is cut off, and the result output is reflected by measuring the fluorescence intensity of AMC. The implementation of the adder is to fix two or more enzymes in a CSTR, input the polypeptide chains corresponding to each enzyme for reaction at the same time, detect the fluorescence intensity of AMC, and produce AMC fluorescence intensity equal to the sum of the single fluorescence intensity produced by each enzyme

reaction. Subtraction is to fix two enzymes in two CSTRs respectively, then connect them in series. The polypeptide chains corresponding to the two enzymes are the same, carrying two enzyme cleavage sites. After the first CSTR, the polypeptide chain is partially cut, and the uncut polypeptide chain continues to be input into the second CSTR. The fluorescence intensity of the cut AMC is equivalent to the fluorescence intensity of all inputs into the second CSTR minus the fluorescence intensity of the first CSTR. The implementation of multiplication is to fix two enzymes in a CSTR, and then design a polypeptide chain with two cleavage sites. Only when it is cut by two enzymes at the same time can free AMC be generated. By adding two enzyme inhibitors, the activity of the two enzymes can be regulated, and the final AMC fluorescence intensity is equal to the product of the activities of the two enzymes. This research cannot directly observe the results through the output, it needs to measure the activity of a single enzyme in advance, and also strictly control the consistency of the substrate concentration.

The half adder, half subtractor, and arithmetic operation device built based on the split pool pave the way for arithmetic calculations using enzymes, but at the same time, complex experimental operations also limit its expansion and application. The design of enzyme-based cascade systems needs to carefully consider the specificity of enzymes and the cross-reactivity between substrates to ensure the accuracy and reliability of the system.

12.4 Solving NP-complete Problems Based on Protein Molecules

Actin and tubulin proteins are important components of the cytoskeleton and participate in cell transport functions. They obtain energy through ATP hydrolysis and can move quickly along the cell skeleton (actin or tubulin) under the drive of actin coagulation protein and driving protein. Actin and tubulin proteins have the characteristics of low price, spontaneous propulsion, independent operation without mutual influence, small size, fast movement speed, and unidirectional movement, which can be used for protein calculation.

Nicolau and others proposed a method to solve the subset sum problem using actin and tubulin proteins [2]. The subset sum problem is a classic NP-complete problem. Its goal is to determine whether there is a subset in a given set whose sum of elements equals a given target value. More specifically, for a set S containing N integers = s1, s2, …, sN and a target sum T, the subset sum problem requires determining whether there is a subset whose sum of elements equals T. Nicolau and others set the set to 2,5,9. In order to find all the subset sums of this set, they first used electron beam lithography to etch the network on a silicon dioxide substrate according to the following rules. According to the numbers in the set, 3 batches of intersections are etched in turn, etching 2, 5, and 9 rows respectively; the first row of each batch etches a splitting intersection, and the remaining rows etch

through intersections; all rows of intersections do not divide batches, the first row etches one intersection, and each subsequent row increases one intersection; except for the one intersection in the first row that only has one input channel and two output channels, other nodes all have two input channels and two output channels; channels are etched between the output and input channels of the intersections of adjacent two rows to connect them; the total output of the last row is all possible results 0–16. The splitting intersection is a node that can move in two directions, and the through intersection is a node that can only go straight. Actin and tubulin proteins are injected into the upper left corner of the network, and a buffer solution containing ATP is provided at the same time. Actin and tubulin proteins will move quickly in the network under the drive of actin coagulation protein and driving protein, and only come out from the correct result. The action path of the protein can be obtained through fluorescence microscopy, and the correct subset sum result can be obtained.

Nicolau [34] believes that if specific NP-complete problems can be graphically represented, then these problems can be transformed into the design of physical networks, such as the design of channels, nodes, entrances, and exits, and other microfluidic structures. By encoding the computational network of the NP-complete problem of interest, the problem is solved in parallel by the random exploration of a large number of independent computational substances. These computational substances are non-biological and biological. In non-biological substances, laminar flow fluids and micrometer-sized non-biological beads have been used to solve NP-complete problems, but non-biological substances often have the limitation of a shorter moving life. Biological substances used to explore the network include cytoskeletal filaments (actin filaments or microtubules), prokaryotes (bacteria and archaea), eukaryotes, etc., but only cytoskeletal filaments (actin filaments or microtubules) have been experimentally verified.

12.5 Protein Storage

Traditional storage media are mainly based on silicon chips or magnetic materials, but these methods have some limitations, such as limited storage density, high power consumption, and susceptibility to magnetic fields and radiation interference. To overcome these problems, researchers have begun to explore new storage technologies, one of which is protein storage technology. Protein storage technology originates from the study of bacteriorhodopsin, which is photosensitive. When excited by light of a specific wavelength, it undergoes structural changes and maintains a stable state, making it one of the candidate materials for storing and reading data. With the emergence of the emerging non-volatile memory resistors, recent research on protein storage has focused on protein-based memory resistors, such as silk protein, ferritin, and egg protein, which are introduced below.

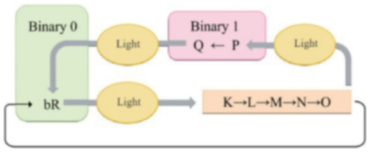


Fig. 12.12 The BR light cycle includes the ground state (bR) and intermediate states K, L, M, N, and O

12.5.1 Protein Storage Based on Bacteriorhodopsin

Bacteriorhodopsin (BR) is a protein found in the cell membrane of halophilic bacteria, belonging to the G-protein coupled receptor (GCPR) family, composed of seven alpha helices. BR can pump protons out of the membrane under light drive, this unique photoelectric response characteristic makes it one of the biological materials with great application potential [35]. As shown in Fig. 12.12, when BR is exposed to light, it undergoes structural changes in a certain order, such as the ground state (bR) and intermediate states K, L, M, N, O [36]. When excited by green light, the BR molecule is excited from the ground state to the K state, and relaxes back to the O state. If the BR molecule receives red light excitation higher than the O state energy, it will convert to the P state, and then gradually decay to the long-lived intermediate state Q (>5 years). Using the two different states of bR and Q during the structural change process can be used to represent digital information 0 and 1. The Kock Center for Molecular Electronics in the United States has made a model of this storage system, which is a transparent container, 1x1x2 inches in size, filled with polyacrylamide gel, and BR is placed in it, forming a three-dimensional data array. When the protein is not excited, it is in the bR state and is fixed in a certain place by the polymerization with the gel. Around the container, there is a set of krypton lasers and charge injection devices (CID) displayed for reading and writing data [37].

To write data, first use a "page" laser beam to excite the protein molecules, making them go from the bR state to the O state. By controlling the page laser beam, it can only excite a two-dimensional plane of the material in the container, and this excited plane in the O state can write data. This process is called "paging", and this page can store 4096×4096 bits of data. After paging, you must complete the write data operation with a red "write data" laser beam before the material returns from the O state to the bR state. The protein molecules storing data 1 are irradiated by the write laser beam, and they further transition from the O state to the Q state, while the protein molecules storing data 0 are not irradiated by the write data laser beam, and quickly return from the O state to the bR state. Using the write data device, the data to be written can be written into the protein storage in pages.

To read data, first use a page laser beam to excite the target page into the O state. The purpose of doing this is to further enlarge the difference in absorption spectrum between the bR and Q states, making the read data less confusing. Then use the read laser beam and the charge injection device CID array to represent (read) the data of the target page in the form of an image. To erase a page of data, you can use a short-pulse blue laser beam to irradiate the page, causing the Q state molecules

to return to the bR state. If you want to erase all the data in the container, you can expose the container under an incandescent lamp with ultraviolet output. To ensure data accuracy, two additional parity bits can be used to correct errors during read and write operations.

BR protein has been used in some preliminary applications in the field of information storage. Compared with semiconductor storage technology, protein storage technology has many advantages: very fast data access speed, large storage capacity, stable and reliable operation, stable data preservation, can be mass-produced using genetic engineering, and is cheap. Although BR has these advantages in the field of protein storage, there are still challenges and limitations. For example, better control and optimization of the writing and reading process are needed to improve the stability and reliability of data. In addition, technical problems related to material preparation, integrated optical systems, and data decoding need to be solved to realize the commercialization and feasibility of bacteriorhodopsin in practical applications.

12.5.2 Protein-Based Memory Resistors

Memristor (also known as resistive memory) is the fourth type of passive electronic component after resistors, capacitors, and inductors. It is an emerging non-volatile memory and is considered an ideal alternative to traditional memory. It shows great application potential in fields such as artificial intelligence, neuromorphic computing, and high-density data storage [38]. A typical memristor has a metal-insulator-metal (MIM) structure, including two electrodes and an active layer. The active layer is the main functional layer of the memristor. Various materials have been developed as active components of memristors, including binary and multivariate oxides, organic polymer materials, and sulfur compounds [39]. However, these materials may produce non-degradable and toxic electronic waste during production and after disposal, causing environmental problems. Protein materials, due to their good biocompatibility, controllable biodegradability, and remarkable mechanical toughness, have attracted widespread attention and are considered ideal materials for developing high-performance green flexible electronic components [40]. In addition, protein-based memristors are truly biomaterials that can be used to create biomimetic neural network electronic components. They have application prospects in implantable computing and direct human-machine interaction and integration that existing inorganic components cannot match.

The amino acid residues in protein materials have excellent ion binding ability, which helps to form conductive filament paths under the action of an electric field, making most proteins exhibit resistive switching characteristics. Silk protein, ferritin, egg protein, and sericin protein have been confirmed to have typical resistive switching characteristics through research, making them potentially valuable in the field of data storage and arousing widespread research interest. Figure 12.13 is the development history of protein-based memristors, which has gone through simple

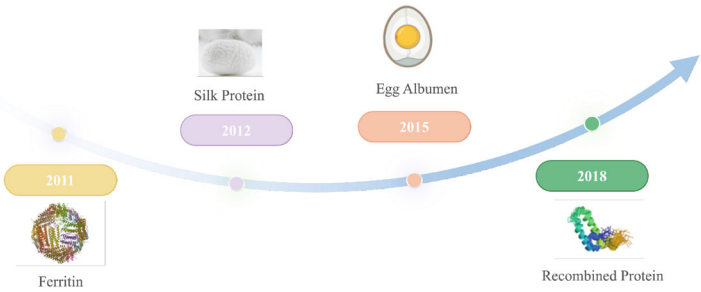


Fig. 12.13 Development history of protein-based memristors

tests of natural proteins, research on the optimization of protein-based memristors by functional assembly, and exploration in the field of artificially designed recombinant proteins. Before formally introducing protein-based memristors, let's first introduce two important indicators for evaluating memristor performance: switch ratio and retention time. The switch ratio (ON/OFF ratio) is the ratio of resistance or current between the ON and OFF states of the memristor. The switch ratio is one of the important indicators to measure the performance of the memristor. A higher switch ratio means that the memristor can significantly change the resistance or current value when switching states, which is crucial for accurately reading and storing information. The retention time is the length of time that the memristor can maintain a certain state after switching to that state. Retention time is also an important indicator for evaluating memristor performance. It measures the stability and durability of the memristor's stored information. A longer retention time means that the memristor can maintain the stored information for a longer time, while a shorter retention time may lead to information loss or instability.

12.5.2.1 Ferritin Memristor

Ferritin is a major intracellular iron storage protein, a spherical metalloprotein with a diameter of about 12 nm, composed of a shell about 2 nm thick and an iron storage cavity with a diameter of about 8 nm. Ferritin has high stability and can maintain its structure and function under a wide pH range (2.0–12.0) and high temperature (up to 80 °C). This stability makes ferritin an ideal natural biomaterial. In addition, iron ions can be released from the shell under the action of an electric field, which may be the reason for the observable resistive switching effect [41].

In 2011, the Cho research group reported a memristor constructed using ferritin, clarifying that the reversible resistance change in the ferritin nanoparticle film may be caused by the charge capture/release of the Fe3+/Fe2+ redox pair, and proved that ferritin nanoparticles can serve as nanoscale memory devices. This layer-by-layer assembled protein multilayer device can be extended to biomimetic electronic devices at the molecular level with adjustable memory performance [42]. The

Chen research group also explored ferritin-based memristors, integrating ferritin into precise nanogaps generated by chemical methods to achieve resistive memory behavior, and explored the regulation of memory device performance by adjusting iron content [41, 43]. This regulation process is attributed to the high redox activity of the inorganic iron chelate core structure under the action of high concentration iron. The greater the iron load, the better the memory performance. The high load of iron leads to the release of more iron from the ferritin core, resulting in the transport of iron (II) ions to the outside of the protein cage. Zhang and others constructed a typical Pt/ferritin/Pt MIM structure memristor on a Si/SiO_2 substrate, where the protein film thickness reached 250 nm, the device's turn-on voltage was 1.3 V, and the turn-off voltage was −0.4 V [44]. Since the electrode used is chemically inert platinum, it is speculated that the conductive filament is composed of iron ions.

12.5.2.2 Silk Protein Memristor

Silk protein (Fibroin, also known as sericin protein) is a natural high-molecular-weight fibrous protein extracted from silk, accounting for about 70–80% of silk. It is composed of 18 amino acids, among which glycine, serine, and alanine account for more than 80% of the total. The basic unit of silk protein consists of a heavy chain, a light chain, and a glycoprotein P25. The heavy chain and the light chain are connected by disulfide bonds, and then combined with glycoprotein P25 through non-covalent interactions such as hydrophobic bonds. The heavy chain is the main component of silk protein, containing N-terminal and C-terminal hydrophilic structural domains, as well as 12 highly repetitive regions rich in Gly-Ala. The N-terminal of the heavy chain is a significant two-layer entangled β-fold structure [45].

Silk protein has attracted much attention due to its excellent biocompatibility, optical transparency, ultra-light weight, superior flexibility, and mechanical properties. At present, a large amount of research has been conducted on memory resistors based on silk protein. In 2012, the Hota research group used silk protein to construct a typical MIM structure memory resistor, observed bipolar memory switching behavior for the first time, and confirmed that its switching mechanism is due to the switching of conductive filaments under the ion effect [46]. In 2015, the Chen research group at Nanyang Technological University first reported a configurable memory resistor based on silk protein. This memory resistor also uses a simple MIM structure, with gold (Au) as the bottom electrode and silver (Ag) as the top electrode [47]. By controlling the compliance current during the setting process, the resistance switching (RS) type of the device can be precisely controlled. A higher compliance current ($>100\,\mu A$) can trigger storage RS, while a lower compliance current ($<10\,\mu A$) can trigger threshold-type RS memory, with a switching ratio of up to 10^7 and a retention time of more than 4500 s. In 2016, the research group used silk protein as both the active layer and the substrate, with metal (Mg) as the upper and lower electrodes, and prepared a silk protein (substrate)/Au/Mg/silk protein (switching layer)/Mg structure physical transient

RRAM, showing reasonable bipolar memory characteristics, with a switching ratio of 10^2 and a retention time of more than 10^4 [48]. These devices can be completely dissolved in deionized water (DI) or phosphate-buffered saline (PBS, pH 7.4) within 2 hours, proving that the proposed transient storage devices based on silk protein have great application potential in safe data storage systems, biocompatible and implantable electronic devices. The structure of the ultra-light memory resistor reported by the Chen research group later is silk protein (substrate)/Au/silk protein (switching layer)/Ag, with a unit area mass of only $0.4\,\mathrm{mg\,cm^{-2}}$, which is much lighter than traditional silicon substrates or office paper, 320 times lighter than traditional silicon substrates, 20 times lighter than office paper, can be maintained by a single hair, with a switching ratio of 10^5 and a retention time of about 10^4 s [49].

The Liu research group found that silk protein forms a nanocrystalline network with a fishnet-like topological structure [38, 50], which is connected by β-microcrystalline phases formed by orderly arranged β-fold structures within the molecule. The unique mesoscopic network structure of silk protein gives it great potential for application modification. By doping some functional elements (such as nanometal clusters, quantum dots or conductive polymers), it can give silk protein materials additional properties without affecting their original properties. In 2013, the Gogurla team doped gold nanoparticles (Au) into silk protein. NPs) are integrated into silk protein biopolymers, constructing a silk protein-based memristor, and improving the switch (ON/OFF) ratio of working voltage and current, with a switch ratio of up to 10^6, and the on/off voltage concentrated at 2 V/−2 V [51]. This improvement is attributed to the synergistic effect of negatively charged gold nanoparticles and positively charged oxidized silk proteins, enhancing the conductive path in the switch layer. In 2019, the Liu research group used bovine serum protein (BSA) as a carrier, doped silver nanoclusters into silk protein materials, promoted the crystallization process of silk protein molecules, formed a stable functional structure, realized the mesoscopic functionalization of silk protein materials, and used it as a resistive material layer to construct a new type of silk protein-based memristor [52]. Compared with the non-functionalized silk protein memristor, the performance of the protein-based mesoscopic memristor significantly improved after functionalization, the switching speed ($\approx$10 ns) significantly increased, the switching stability was very good, with ultra-low switching voltage (the on and off voltages fluctuated around 0.30 and −0.18 V respectively), thus greatly reducing power consumption. In addition to functional doping of metal nanoclusters, other light-responsive functional materials, such as carbon dots, can also be selected. Biocompatible carbon dots (cd) with small size, good optical properties, and low production cost have become important materials in the field of optoelectronics. The Han research group prepared a light-adjustable memristor by doping carbon nanodots into a silk protein matrix. Taking different top electrodes (Al, Au, and Ag) as examples, they systematically studied the characteristics of the new structure memory device based on the metal top electrode/carbon dot-silk protein/indium tin oxide (ITO), and found that carbon dots with light-adjustable

charge capture ability play an important role in the light-adjustable resistance switching characteristics of the memristor [53].

12.5.2.3 Egg Albumen Memristor

Egg albumen (EA) is a common and easily obtainable natural protein, with good biocompatibility, biodegradability, flexibility, and low cost, widely used to construct protein-based memristors. Egg albumen consists of about 10% proteins (such as albumin, mucin, and globulin) and 90% water. Most of the proteins in egg albumen are globular proteins, connected by many weak chemical bonds. During the process of processing protein materials into thin films, that is, during the heat baking process, weak bonds break, and the two main chemical bonds of protein molecules, peptide bonds and disulfide bonds, cross-link. The formation of disulfide bonds is an irreversible process (called coagulation), which is the reason for the generation of heat-crosslinked solid protein films [54]. Egg albumen has a rich amino acid composition, which includes polar groups such as hydroxyl, carboxyl, and amino groups. These groups can change their charge state and molecular structure under the action of an electric field, leading to resistance changes and realizing the memristive effect. Egg albumen can be directly extracted from fresh eggs without the need for additional chemical purification or extraction, reducing manufacturing costs and method complexity.

In 2015, Chen et al. used egg protein directly obtained from fresh eggs to prepare a typical ITO/egg protein/Al MIM structure memristor without any additional purification or extraction. The device performed excellently, with a switching ratio reaching 10^3 and a retention time exceeding 10^4 s, confirming that the formation and rupture of conductive filaments were due to the electric field-induced migration of oxygen ions and the electrochemical oxidation-reduction reaction of iron ions [55]. He et al. used egg protein as the active layer and water-soluble Mg and tungsten (W) as the upper and lower electrodes to prepare a transient memristor device, which has better reliability and stability, proving the possibility of information storage in transient electronic devices and opening the door for the manufacture of biocompatible, biodegradable electronic devices using cheap, abundant, and natural bioelectronic application materials [54]. In 2017, Zhu et al. used Ag and ITO as the upper and lower electrodes to construct a new type of egg protein-based memristor, which has a long retention time (10^4 s), a faster device erasure speed (75 ns), and its switching voltage is also lower, respectively maintained at 0.6 V/−0.7 V [56]. Subsequently, Yan's team prepared a new type of memristor with a W/egg protein/ITO/polyethylene terephthalate (PET) structure. The device works normally under mechanical bending conditions, with no significant performance degradation, demonstrating good flexible resistive storage performance. The device can also achieve synaptic functions by adjusting the applied pulse voltage [57]. In 2019, Zhou et al. made a super-flexible egg protein memristor array, achieving multi-functional logic gates including "AND", "OR" and "NOT" operations. It is worth noting that both the active layer and the flexible substrate are made of egg protein

through physical and chemical processing [58]. By simultaneously or independently applying three separate signals (including two different electrical pulses and one broadband light pulse) to the memristor, it is easy to switch logic states between the three basic logic operations.

12.5.2.4 Other Protein-Based Memristors

In addition to these representative proteins, some other proteins are also used to construct memristors, such as sericin, a by-product of silk protein, proteins extracted from microorganisms, and artificially recombined proteins.

Sericin, as a by-product of silk processing, has also been proven to have excellent memristive switching performance. Sericin is composed of 18 amino acids, most of which have strong polar groups, such as hydroxyl, carboxyl, amino, etc., and is a stable, non-easily oxidized, water-soluble, biodegradable material. Secondly, sericin can form a uniform and dense film, which has insulating properties in its natural state, suitable for manufacturing storage devices. Sericin is also an easily obtainable abundant material, which is usually a by-product discarded during silk processing, so its cost is also very low. In 2013, Chen's team used sericin to construct a memristor, with Ag and Au as the upper and lower electrodes, and its switching ratio can reach 10^6, with a retention time exceeding 10^3 s [59]. By using different limiting currents to realize multi-level storage of resistive memory through the charge carrier capture/release mechanism, the research confirmed the feasibility of natural material waste by-products in the field of data storage, but further research is still needed to improve the stability of the device.

S-layer (surface layer, Slp) is a two-dimensional biomembrane structure formed by the repeated arrangement of single-molecule proteins or glycoproteins, which exists on the outermost layer of bacterial and archaeal cell walls, commonly found in lactic acid bacteria, such as acidophilus, chicken lactobacillus, cheese lactobacillus, brucella lactobacillus, bulgarian lactobacillus, kefir lactobacillus, and short lactobacillus, etc. The molecular weight of S-layer protein is between 40–200 kDa, and it has a highly porous and irregular lattice structure at the sub-nanometer level (about 10 nm thick protein mesh structure), can recrystallize in vitro and fuse with exogenous proteins, thus becoming a good candidate material for applications in bionanotechnology and biomimetics. In 2018, Moudgil et al. first proposed a flexible storage device constructed from S-layer protein (Al/S-layer protein/ITO/PET), which can switch to low resistance state (LRS) and high resistance state (HRS) in a bistable manner, has stable bistable memory performance, has a longer retention time ($>4 \times 10^3$ s), can withstand 500 cycle tests, and the device can withstand more than 100 bending resistive performance tests, meeting the expected applicability of biocompatible wearable electronic devices [60].

Lee and others used heat-denatured protein hexa-His tagged recombinant molecular chaperone protein DnaJ (rDnaJ) as an active layer to create a bio-memristor [61]. The rDnaJ memristor demonstrated extremely low device turn-on voltage (about 0.12 V) and reset voltage (about −0.08 V), as well as a high switching ratio

($>10^6$) and a long performance retention time ($>10^6$ s). The heat-denatured rDnaJ protein layer between the metal electrodes can control the formation/breakage of copper conductive filaments by adjusting the metal chelating ability of the amino acid residues in the protein, achieving high-performance non-volatile resistive memory performance. This study uses recombinant proteins with engineered properties as powerful building blocks to meet the requirements of next-generation biocompatible, flexible, high-performance, and low-power electronic products. However, current research on how to regulate device performance through recombinant proteins is very limited, mainly for the following reasons. First, recombinant DNA technology is more often used in treatment and diagnosis, and how to apply this advanced and complex technology to the field of data storage still requires more in-depth interdisciplinary communication. Second, for some metal proteins, recombinant DNA technology may pose certain challenges in modifying their structure, which also limits the scope and application of protein recombination regulation device performance research. In addition, unexpected structural changes may occur in the traditional device manufacturing process, which may cause changes in the expected properties of recombinant proteins. Finally, a full understanding of the relationship between hierarchical structure and device performance is crucial for more advanced material and device design, and further research is needed to develop design strategies from genetic engineering to target device functions [39].

References

1. Bray D.: Protein molecules as computational elements in living cells. Nature **376**(6538), 307–312 (1995).
2. Nicolau Jr D V, Lard M, Korten T, et al.: Parallel computation with molecular-motor-propelled agents in nanofabricated networks. Proceedings of the National Academy of Sciences **113**(10), 2591–2596 (2016).
3. Gao X J, Chong L S, Kim M S, et al.: Programmable protein circuits in living cells. Science **361**(6408), 1252–1258 (2018).
4. Fink T, Lonzarić J, Praznik A, et al.: Design of fast proteolysis-based signaling and logic circuits in mammalian cells. Nature chemical biology **15**(2), 115–122 (2019).
5. Chen Z, Kibler R D, Hunt A, et al.: De novo design of protein logic gates. Science **368**(6486), 78–84 (2020).
6. Dueber J E, Yeh B J, Chak K, et al.: Reprogramming control of an allosteric signaling switch through modular recombination. Science **301**(5641), 1904–1908 (2003).
7. Wodak S J, Paci E, Dokholyan N V, et al.: Allostery in its many disguises: from theory to applications. Structure **27**(4), 566–578 (2019).
8. Dokholyan N V.: Nanoscale programming of cellular and physiological phenotypes: inorganic meets organic programming. NPJ systems biology and applications **7**(1), 15 (2021).
9. Chen J, Vishweshwaraiah Y L, Mailman R B, et al.: A noncommutative combinatorial protein logic circuit controls cell orientation in nanoenvironments. Science Advances **9**(21), eadg1062 (2023).
10. Vishweshwaraiah Y L, Chen J, Chirasani V R, et al.: Two-input protein logic gate for computation in living cells. Nature communications **12**(1), 6615 (2021).
11. Fratto B E, Lewer J M, Katz E.: An Enzyme Based Half-Adder and Half-Subtractor with a Modular Design. ChemPhysChem **17**(14), 2210–2217 (2016).

12. Katz E, Privman V.: Enzyme-based logic systems for information processing. Chemical Society Reviews **39**(5), 1835–1857 (2010).
13. Baron R, Lioubashevski O, Katz E, et al.: Logic gates and elementary computing by enzymes. The Journal of Physical Chemistry A **110**(27), 8548–8553 (2006).
14. Strack G, Pita M, Ornatska M, et al.: Boolean logic gates that use enzymes as input signals. ChemBioChem **9**(8), 1260–1266 (2008).
15. Katz E.: Boolean Logic Gates Realized with Enzyme-catalyzed Reactions–Unusual Look at Usual Chemical Reactions. ChemPhysChem **20**(1), 9–22 (2019).
16. Baron R, Lioubashevski O, Katz E, et al.: Two coupled enzymes perform in parallel the 'AND'and 'InhibAND' logic gate operations. Organic and biomolecular chemistry **4**(6), 989–991 (2006).
17. Chuang M C, Windmiller J R, Santhosh P, et al.: High-fidelity determination of security threats via a Boolean biocatalytic cascade. Chemical Communications **47**(11), 3087–3089 (2011).
18. Halámek J, Bocharova V, Arugula M A, et al.: Realization and properties of biochemical-computing biocatalytic XOR gate based on enzyme inhibition by a substrate. The Journal of Physical Chemistry B **115**(32), 9838–9845 (2011).
19. Privman V, Zhou J, Halámek J, et al.: Realization and properties of biochemical-computing biocatalytic XOR gate based on signal change. The Journal of Physical Chemistry B **114**(42), 13601–13608 (2010).
20. Filipov Y, Domanskyi S, Wood M L, et al.: Experimental Realization of a High-Quality Biochemical XOR Gate. ChemPhysChem **18**(20), 2908–2915 (2017).
21. Fratto B E, Roby L J, Guz N, et al.: Enzyme-based logic gates switchable between OR, NXOR and NAND Boolean operations realized in a flow system. Chemical Communications **50**(81), 12043–12046 (2014).
22. Zhou J, Arugula M A, Halamek J, et al.: Enzyme-based NAND and NOR logic gates with modular design. The Journal of Physical Chemistry B **113**(49), 16065–16070 (2009).
23. Niazov T, Baron R, Katz E, et al.: Concatenated logic gates using four coupled biocatalysts operating in series. Proceedings of the National Academy of Sciences **103**(46), 17160–17163 (2006).
24. de Ronde W, ten Wolde P R, Mugler A.: Protein logic: a statistical mechanical study of signal integration at the single-molecule level. Biophysical Journal **103**(5), 1097–1107 (2012).
25. Gunnoo S B, Finney H M, Baker T S, et al.: Creation of a gated antibody as a conditionally functional synthetic protein. Nature Communications **5**(1), 4388 (2014).
26. Oostindie S C, Rinaldi D A, Zom G G, et al.: Logic-gated antibody pairs that selectively act on cells co-expressing two antigens. Nature Biotechnology **40**(10), 1509–1519 (2022).
27. Moon T S, Lou C, Tamsir A, et al.: Genetic programs constructed from layered logic gates in single cells. Nature **491**(7423), 249–253 (2012).
28. Omersa N, Aden S, Kisovec M, et al.: Design of protein logic gate system operating on lipid membranes. ACS synthetic biology **9**(2), 316–328 (2020).
29. McCue A C, Kuhlman B.: Design and engineering of light-sensitive protein switches. Current opinion in structural biology **74**, 102377 (2022).
30. Unger R, Moult J.: Towards computing with proteins. Proteins: Structure, Function, and Bioinformatics **63**(1), 53–64 (2006).
31. Bordoy A E, O'Connor N J, Chatterjee A.: Construction of two-input logic gates using transcriptional interference. ACS Synthetic Biology **8**(10), 2428–2441 (2019).
32. Siuti P, Yazbek J, Lu T K.: Synthetic circuits integrating logic and memory in living cells. Nature biotechnology **31**(5), 448–452 (2013).
33. Ivanov N M, Baltussen M G, Regueiro C L F, et al.: Computing arithmetic functions using immobilised enzymatic reaction networks. Angewandte Chemie **135**(7), e202215759 (2023).
34. van Delft F C, Ipolitti G, Nicolau Jr D V, et al.: Something has to give: scaling combinatorial computing by biological agents exploring physical networks encoding NP-complete problems. Interface focus **8**(6), 20180034 (2018).
35. Feng H. Future storage technology–protein storage. Recording media technology (5), 48–51 (2008).

36. Stuart J A, Marcy D L, Wise K J, et al.: Volumetric optical memory based on bacteriorhodopsin. Synthetic metals **127**(1–3), 3–15 (2002).
37. Chen Y S. Protein storage will replace semiconductor storage. World Science (11), 28–30 (1996).
38. Shi C Y, MIN G Z, Liu X Y. Research progress of protein-based memristor. Acta physica Sinica (2020).
39. Wang J, Qian F, Huang S, et al.: Recent Progress of Protein-Based Data Storage and Neuromorphic Devices. Advanced Intelligent Systems **3**(1), 2000180 (2021).
40. Zhu B, Wang H, Leow W R, et al.: Silk fibroin for flexible electronic devices. Advanced Materials **28**(22), 4250–4265 (2016).
41. Meng F, Sana B, Li Y, et al.: Bioengineered tunable memristor based on protein nanocage. Small (Weinheim an der Bergstrasse, Germany) **10**(2), 277–283 (2014).
42. Ko Y, Kim Y, Baek H, et al.: Electrically bistable properties of layer-by-layer assembled multilayers based on protein nanoparticles. ACS nano **5**(12), 9918–9926 (2011).
43. Meng F, Jiang L, Zheng K, et al.: Protein-based memristive nanodevices. Small **7**(21), 3016–3020 (2011).
44. Zhang C, Shang J, Xue W, et al.: Convertible resistive switching characteristics between memory switching and threshold switching in a single ferritin-based memristor. Chemical communications **52**(26), 4828–4831 (2016).
45. He Y X, Zhang N N, Li W F, et al.: N-terminal domain of Bombyx mori fibroin mediates the assembly of silk in response to pH decrease. Journal of molecular biology **418**(3–4), 197–207 (2012).
46. Hota M K, Bera M K, Kundu B, et al.: A natural silk fibroin protein-based transparent bio-memristor. Advanced Functional Materials **22**(21), 4493–4499 (2012).
47. Wang H, Du Y, Li Y, et al.: Configurable resistive switching between memory and threshold characteristics for protein-based devices. Advanced Functional Materials **25**(25), 3825–3831 (2015).
48. Wang H, Zhu B, Ma X, et al.: Physically Transient Resistive Switching Memory Based on Silk Protein. Small (Weinheim an der Bergstrasse, Germany) **12**(20), 2715–2719 (2016).
49. Wang H, Zhu B, Ma X, et al.: Ultra-Lightweight Resistive Switching Memory Devices Based on Silk Fibroin. Small (Weinheim an der Bergstrasse, Germany) **12**(25), 3360–3365 (2016).
50. Song Y, Lin Z, Kong L, et al.: Meso-Functionalization of Silk Fibroin by Upconversion Fluorescence and Near Infrared In Vivo Biosensing. Advanced Functional Materials **27**(26), 1700628 (2017).
51. Gogurla N, Mondal S P, Sinha A K, et al.: Transparent and flexible resistive switching memory devices with a very high ON/OFF ratio using gold nanoparticles embedded in a silk protein matrix. Nanotechnology **24**(34), 345202 (2013).
52. Shi C, Wang J, Sushko M L, et al.: Silk flexible electronics: from Bombyx mori silk Ag nanoclusters hybrid materials to mesoscopic memristors and synaptic emulators. Advanced Functional Materials **29**(42), 1904777 (2019).
53. Lv Z, Wang Y, Chen Z, et al.: Phototunable biomemory based on light-mediated charge trap. Advanced Science **5**(9), 1800714 (2018).
54. He X, Zhang J, Wang W, et al.: Transient resistive switching devices made from egg albumen dielectrics and dissolvable electrodes. ACS applied materials and interfaces **8**(17), 10954–10960 (2016).
55. Chen Y C, Yu H C, Huang C Y, et al.: Nonvolatile bio-memristor fabricated with egg albumen film. Scientific reports **5**(1), 10022 (2015).
56. Zhu J X, Zhou W L, Wang Z Q, et al.: Flexible, transferable and conformal egg albumen based resistive switching memory devices. RSC advances **7**(51), 32114–32119 (2017).
57. Yan X, Li X, Zhou Z, et al.: Flexible transparent organic artificial synapse based on the tungsten/egg albumen/indium tin oxide/polyethylene terephthalate memristor. ACS applied materials and interfaces **11**(20), 18654–18661 (2019).
58. Zhou G, Ren Z, Wang L, et al.: Artificial and wearable albumen protein memristor arrays with integrated memory logic gate functionality. Materials Horizons **6**(9), 1877–1882 (2019).

59. Wang H, Meng F, Cai Y, et al.: Sericin for resistance switching device with multilevel nonvolatile memory. Advanced Materials (Deerfield Beach, Fla.) **25**(38), 5498–5503 (2013).
60. Moudgil A, Kalyani N, Sinsinbar G, et al.: S-layer protein for resistive switching and flexible nonvolatile memory device. ACS applied materials and interfaces **10**(5), 4866–4873 (2018).
61. Jang S K, Kim S, Salman M S, et al.: Harnessing recombinant DnaJ protein as reversible metal chelator for a high-performance resistive switching device. Chemistry of Materials **30**(3), 781–788 (2018).