Harish Chandran, Nikhil Gopalkrishnan, Sudhanshu Garg, and John Reif, Biomolecular Computing Systems - From Logic Systems to Smart Sensors and Actuators, Invited Chapter 11, Molecular and Biomolecular Information Processing (Editor: Evgeny Katz), Published by Wiley-VCH, Verlag GmbH & Comp. KGaA, Weinheim, Germany, pp. 199-224, July 2012.
ISBN 978-3-527-33228-1 Online ISBN: 9783527645480
DOI: 10.1002/9783527645480

11
Biomolecular Computing Systems

Harish Chandran, Sudhanshu Garg, Nikhil Gopalkrishnan, and John H. Reif

11.1
Introduction

The field of biomolecular computation started 3.5 billion years ago on earth when the first life forms evolved. What distinguished these simple organisms from a collection of inanimate molecules was the ability to process information, and it is this ability of chemistry-based computation that powers life in its various forms even today. Our attempts at exploiting this rich molecular tool set for computation have just begun. This chapter recounts some of the amazing experiments, starting from 1994, which demonstrated how one could process information and perform computation at the molecular scale using DNA.

11.1.1
Organization of the Chapter

We begin (Section 11.2) by discussing our molecule of choice DNA: why it is ideally suited for nanoscale assembly and computation, its structure, and key reactions. Section 11.3 describes the initial work in DNA computing, including Adleman's experiment of solving a small-scale Hamiltonian path problem via DNA computing and the extensions of that work to solve small-scale instances of other hard search problems. Section 11.4 discusses algorithmic assembly via DNA tiling lattices as well as algorithmic tiling theory based on various abstract models of self-assembly and error correction schemes for tilings. Section 11.5 describes experimental advances in purely hybridization-based computation, including hairpin systems, seesaw gates, as well as speedups using localized reactions. Section 11.6 discusses experimental advances in enzyme-based DNA computing. Section 11.7 discusses DNA reaction networks, including various amplifiers, switches, and oscillators. Section 11.8 concludes the chapter with a discussion of future challenges.

Biomolecular Information Processing: From Logic Systems to Smart Sensors and Actuators,
First Edition. Edited by Evgeny Katz.
© 2012 Wiley-VCH Verlag GmbH & Co. KGaA. Published 2012 by Wiley-VCH Verlag GmbH & Co. KGaA.

11.2 DNA as a Tool for Molecular Programming

DNA systems are relatively easy to design, fairly predictable in their geometric structures, and chemically stable, and have been experimentally implemented in a growing number of laboratories around the world. Abstractions for programming DNA systems and software tools that aid in the design, verification, and simulation of such systems have further expanded the horizons of what are feasible using DNA. Most DNA systems are autonomous: they can execute steps with no external mediation after starting, they are programmable, and the tasks executed can be modified without entirely redesigning the system. In contrast, lipids and carbohydrates are not programmable, the function and structure of proteins are hard to design, and RNA is unstable.

11.2.1 DNA Structure

Single-stranded DNA (ssDNA) is a polymer made from repeating units called *nucleotides*. The nucleotide repeats contain both the segment of the backbone of the molecule, which holds the chain together, and a base. ssDNA has asymmetries along its backbone which gives it a directionality. The asymmetric ends of ssDNA are called the 5′ *and* 3′ *ends*. The four bases found in DNA are adenine (abbreviated A), cytosine (C), guanine (G), and thymine (T) (Figure 11.1).

These bases form the alphabet of DNA; the specific sequence of an ssDNA comprises its information content. In living organisms, DNA does not usually exist as a single molecule, but instead as double-stranded DNA (dsDNA): a pair of oppositely directed ssDNA held together via *complementary base binding*, with A hydrogen-bonded preferentially to T, and C hydrogen-bonded preferentially to G. These two long strands entwine like vines, forming a double helix. As hydrogen bonds are not covalent, they can be broken and rejoined relatively easily, for example, by heating. The two types of base pairs form different numbers of hydrogen bonds, AT forming two hydrogen bonds and GC forming three hydrogen bonds. The association strength of hybridization depends on the sequence of complementary bases, the stability increasing with the length of the DNA sequence, and the GC content. This association strength can be approximated by software packages.

11.2.2 Review of DNA Reactions

We review a few key reactions that allow DNA to execute molecular programs. Toehold-mediated strand displacement is the displacement of a single strand of DNA from a double helix by an incoming strand with a longer complementary region to the template strand. The incoming strand has a toehold, an empty single-stranded region on the template strand complementary to a subsequence of the incoming strand, to which it binds initially. It eventually displaces the outgoing

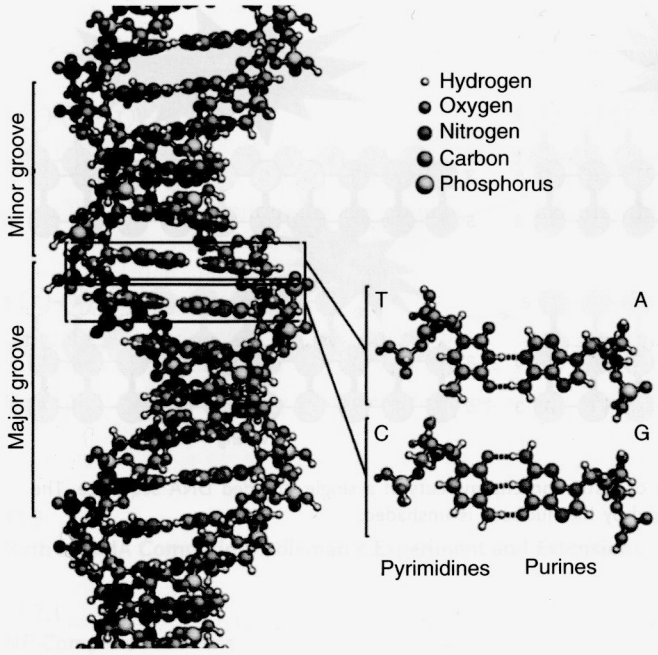

Figure 11.1 Structure of a DNA double helix. (Image by Richard Wheeler, reproduced under the Creative Commons Attribution-Share Alike 3.0 Unported license.)

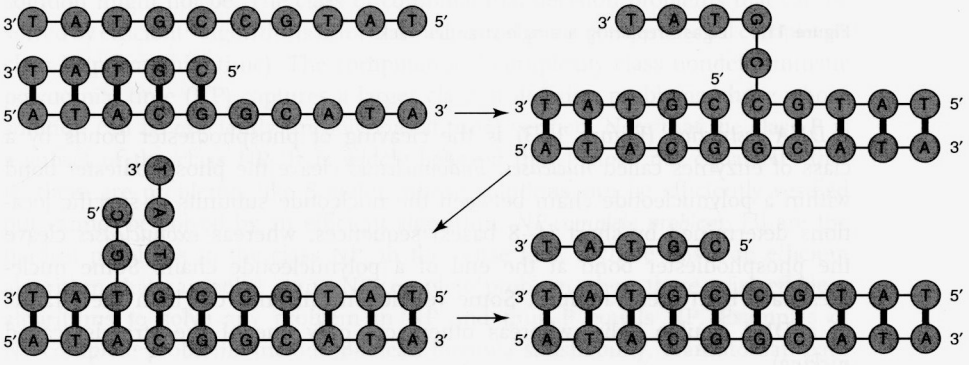

Figure 11.2 Toehold-mediated strand displacement reaction.

strand via a kinetic process modeled as a one-dimensional random walk. Strand displacement is a key process in many of the DNA protocols for running DNA autonomous devices. Toehold exchange is a similar strand displacement reaction mediated by a toehold, with the exception that both the incoming and outgoing strands have distinct, short toeholds on the template strand. Thus, either strand can initiate strand displacement. See [1] for details about the kinetics of the toehold exchange process (Figure 11.2).

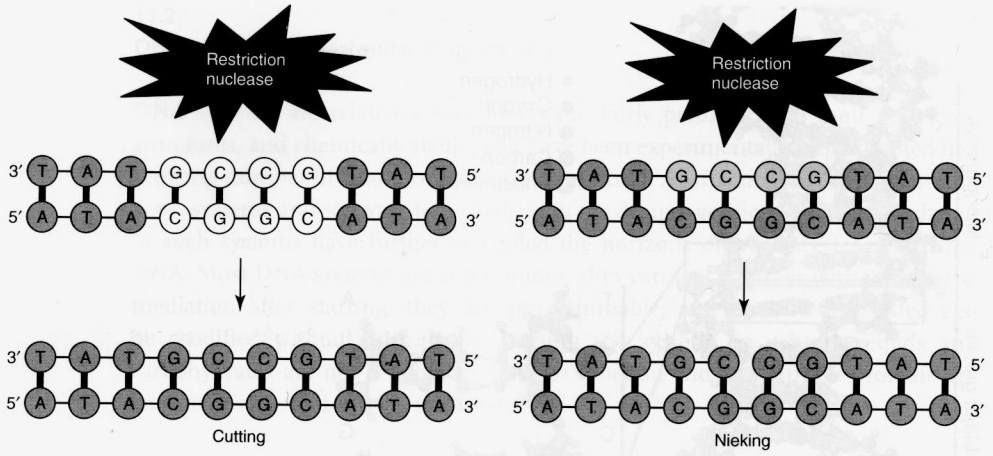

Figure 11.3 Example of restriction enzyme cuts of a single-stranded DNA sequence. The subsequence recognized by the nuclease is unshaded.

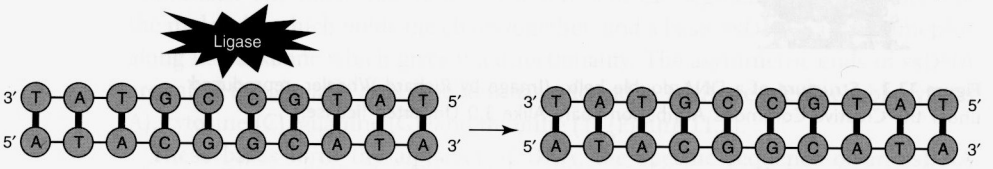

Figure 11.4 Ligase repairing a single-stranded nick.

DNA restriction (Figure 11.3) is the cleaving of phosphodiester bonds by a class of enzymes called *nucleases*. *Endonucleases* cleave the phosphodiester bond within a polynucleotide chain between the nucleotide subunits at specific locations determined by short (4–8 bases) sequences, whereas *exonucleases* cleave the phosphodiester bond at the end of a polynucleotide chain. Some nucleases have both these abilities. Some restriction enzymes cut both the strands of a DNA double helix, whereas others cut only one of the strands (called *nicking*).

DNA ligation (Figure 11.4) is repair of the phosphodiester bond between nucleotides by the class of enzymes known as *ligases*. *Deoxyribozymes* (DNAzymes) are DNA strands that possess enzymatic activity – they can, for example, cleave specific target RNA strands. Typically, they are discovered by *in vivo* evolution search. They have had some use in DNA computations [2]. *DNA polymerases* (Figure 11.5) are a class of enzymes that catalyze the polymerization of nucleoside triphosphates into a DNA strand. The polymerase extends a *primer* strand attached to a DNA template. The newly synthesized sequence is complementary to the template strand.

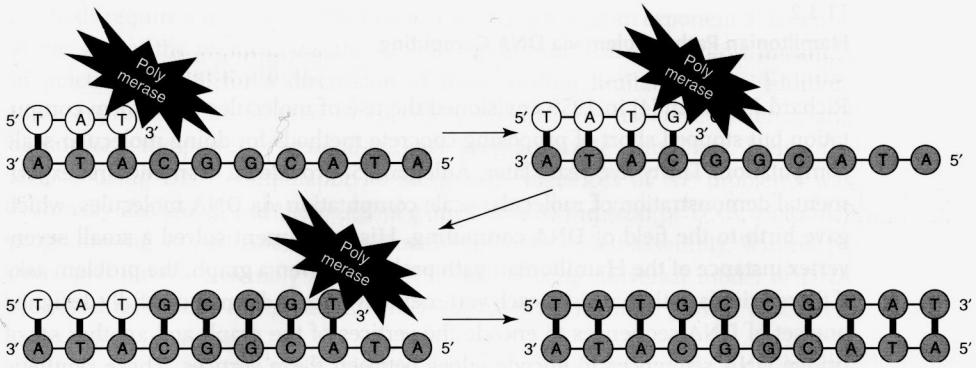

Figure 11.5 DNA synthesis by a polymerase enzyme.

11.3
Birth of DNA Computing: Adleman's Experiment and Extensions

11.3.1
NP-Complete Problems

Anyone who has attempted to solve a Sudoku puzzle would have realized that it is fairly easy to check whether a purported solution is correct, while finding a correct solution might not be. The class of combinatorial decision problems that can be solved by efficient[1] algorithms is formally captured in the computational complexity class P (polynomial time). The computational complexity class nondeterministic polynomial time (NP) captures a larger class of decision problems, those whose solution can be efficiently (in polynomial time) verified. Note that the class P is a subset of the class NP. It is widely believed that P does not equal NP: that is, there are problems, like Sudoku, whose solutions can be efficiently verified but cannot be solved by an efficient algorithm. *NP-complete problems* [3] are the hardest problems in the class NP, in the sense that, if there exists an efficient algorithm that can solve some NP-complete problem, then there exist efficient algorithms to solve any problem in NP and thus P equals NP. Examples of NP-complete problems include Boolean formula satisfiability, Hamiltonian path problem, and Sudoku. Computer scientists were first attracted to DNA computing as a means of solving NP-complete problems, not via efficient algorithms, but rather with the hope of efficiently executing inefficient algorithms in a hugely parallel manner. This hope was belied, but paved the way for DNA molecular computing.

1) Efficient is generally taken to mean that the algorithm's running time is some polynomial function of the size of the problem.

11.3.2
Hamiltonian Path Problem via DNA Computing

Richard Feynman [4] in 1959 envisioned the use of molecules to perform computation but stopped short of proposing concrete methods for doing molecular-scale computation. Thirty-five years later, Adleman [5] provided a dramatic first experimental demonstration of molecular-scale computation via DNA molecules, which gave birth to the field of DNA computing. His experiment solved a small seven-vertex instance of the Hamiltonian path problem. Given a graph, the problem asks if there exists a path that visits each vertex exactly once. Adleman carefully designed one set of DNA sequences to encode the vertices of the graph and another set of bridge DNA sequences to encode edges between these vertices. These synthetic DNA sequences were annealed together in a test tube and formed nanostructures encoding all possible paths in the graph. Note that the huge number of copies of each sequence provides the necessary parallelism to explore this state space. A series of biochemical manipulations were used to isolate DNA nanostructures that encoded any existing Hamiltonian paths. These nanostructures were analyzed to read out the actual path information. The computation was highly energy- and space-efficient (ignoring the energy to isolate the Hamiltonian path), leading to much excitement about its potential uses. However, scientists soon realized that molecular DNA computers could not compete with conventional general-purpose silicon hardware because of issues of error rates, speed, difficulties in reading the output, and so on, which limited the scalability of such methods (Section 11.3.4).

11.3.3
Other Models of DNA Computing

Inspired by Adleman's success, many models were proposed to perform computation with DNA strands. Adleman *et al.* [6] developed a class of methods, known as *sticker-based methods*, for solving Boolean formula satisfiability problems. The sticker-based methods created a combinatorial library of DNA sequences that encoded possible Boolean variable assignments. Then a series of hybridization reactions were employed to select out those DNA sequences whose encoding satisfied all the clauses of the Boolean formula.

Lipton [7] developed a method, termed the *test tube model*, that provided a general-purpose instruction set for DNA computing. The test tube model included various biochemical operations, such as merging test tubes and selecting out from a test tube DNA sequences with specified subsequences. The test tube model was theoretically capable of solving NP search problems in polynomial time.

11.3.4
Shortcomings and Nonscalability of Schemes Using DNA Computation to Solve NP-Complete Problems

Using these approaches and a number of related methods, DNA computation was used to solve a number of relatively small instances of NP problems. These

methods require a number of DNA molecules that grow as an exponential function of the size of the problem instance and hence are not scalable to large instances in practice (see [8] for a discussion of these scaling limitations). In addition, these approaches are tailored to specific problems, are use-once systems, and do not have the power of general-purpose programmable computers. Hence the idea of using DNA computation to solve large instances of NP problems was eventually discarded, but the insights gained seeded the field of DNA molecular programming, in particular the computational power of self-assembly. Winfree [9] developed the tile assembly model (TAM) as a Turing universal model of DNA tile-based molecular computation, which captures algorithmic growth processes, as discussed in Section 11.4.

11.4 Computation Using DNA Tiles

Rapid advancements in experimental DNA self-assembly in conjunction with Winfree's TAM (Section 11.4.1) gave rise to a new computational paradigm, namely, computing using self-assembly [54]. DNA nanostructures can be programmed to form tiles that self-assemble in the test tube to form large lattices as shown by Winfree *et al.* [10] with the DX tile and LaBean *et al.* [11] with the TX tile. The DX and TX tiles have pads that specify their interaction with other tiles. The pads are ssDNA sequences that attach via hybridization to pads with complementary sequences. Mao *et al.* [12] performed a laboratory demonstration of computation via tile assembly using TX tiles. Yan *et al.* [13] performed parallel XOR computation in a test tube using Winfree's DX tile. Other simple computations have also been demonstrated. Yan *et al.* [14] demonstrated the 4×4 tile which had pads in two linearly independent directions in contrast to the earlier DX and TX tiles. This tile assembled into two-dimensional (2D) lattices with square holes. These achievements led to a new field, namely, algorithmic tiling theory.

11.4.1 TAM: an Abstract Model of Self-Assembly

Winfree [9] defined a model of algorithmic self-assembly, the TAM. The model studies unit-sized square tiles that interact via specific glues along their edges. The type of glues along a tile's four edges decides its tile type. A set of tile types specifies a tile assembly program. Growth starts from a specified *seed* tile and continues via a series of single tile additions. The number of copies of each tile type is assumed to be limitless. A tile can be added to an empty position if its interactions with its neighboring tiles are sufficiently strong. All possible maximal assembled structures that are assembled from this process are the output of the tile assembly program. Winfree showed that for every possible Turing machine there exists a tile assembly program that uniquely assembles the complete history of the machine's execution, thus laying a theoretical foundation for a form of DNA-based computation, in

particular, molecular computation via assembly of DNA lattices with tiles in the form of DNA nanostructures.

11.4.2 Algorithmic Assembly via DNA Tiling Lattices

Mao *et al.* [12] demonstrated one of the first examples of using DNA tiles to compute a function. They implemented a cumulative XOR of 4 bits using the TX molecule. Their work is discussed more extensively in Section 11.7. Perhaps the simplest nontrivial tiling 2D construction is the Sierpinksi triangle, a fractal structure. In TAM, a set of tile types performing binary XOR can be used to assemble the Sierpinksi triangle. Rothemund [15] designed plastic tiles (about 1 cm in size) that assemble on a fluid layer via surface tension. The assembly was designed to mimic the Sierpinski triangle pattern. The glue interactions of the tiles were specified by hydrophilic and hydrophobic patches along the tile edges. The system took a long time to assemble (60 h) and was beset by errors. In addition, the size of the tiles made it infeasible to get millions or billions of tiles to assemble.

11.4.2.1 Source of Errors

Implementing an XOR computation requires a tile to add itself to an assembly only if two neighboring glues match. There are competing tile types that have one matching glue and these sometimes add spuriously to the assembly. These incorrect attachments may be unstable at first but may later be stabilized by further "correct" attachments. Such errors are called *cooperative binding errors.* Another major source of errors is spurious nucleation: all assemblies are assumed to arise from a seed assembly. However, there are sets of tiles not containing the seed assembly but form stable assemblies. These assemblies often grow to give partial or even incomplete assemblies. These types of errors are not limited to XOR computation and have to be overcome for most nontrivial algorithmic tiling constructions.

Winfree [17] introduced the kinetic tile assembly model (kTAM), a reversible TAM that models tile dissociation, to study the effect of binding strength and tile concentration on the rate of errors in the assembly. The kTAM predicts that cooperative binding errors may be reduced by growing the assemblies slowly at slightly supersaturated tile concentrations, while nucleation errors may be reduced by providing a wide nucleating assembly. Schulman *et al.* [18] attempted to grow a Sierpinski tiling from DNA tiles by assembling the lattices at close to the melting temperature of cooperative binding and also to seed the assembly using tiles. This strategy was only partially successful since the self-assembly of border tiles proved problematic. Rothemund *et al.* [16] instead used a long ssDNA to serve as a scaffold for the assembly of a row of input tiles (Figure 11.6). This same strategy was used to implement tilings that perform binary counting and copying by Barish *et al.* [19]. This nucleating strategy proved more successful; however, a new kind of error was revealed: facet nucleation errors. Spurious growth occurred at places along the crystal facet where no tiles were designed to stably attach. Fujibayashi *et al.*

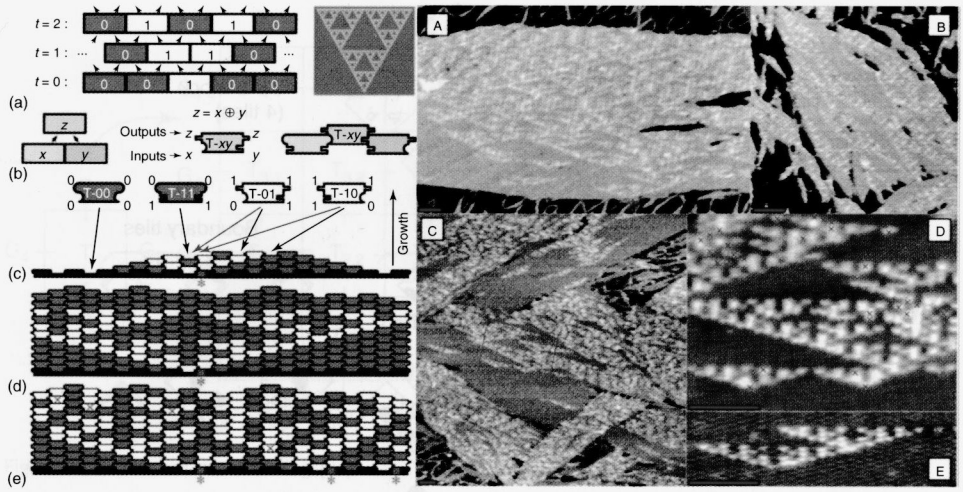

Figure 11.6 Sierpinksi tiling seeded by a long ssDNA [16]. (a) Logical programming of the Sierpinkski triangle. (b) Tiles that achieve this programming. (c), (d) and (e) Self-assembly process. On the right: Atomic force microscopic images of Sierpinksi tiling.

[20] solved this by using border tiles along the facet that did not have sticky ends on their outside, thus preventing any facet nucleation errors. The combination of all these strategies was used by Barish *et al.* [21] to perform binary copying and counting at high yields and low error rates.

11.4.3
Algorithmic Error Correction Schemes for Tilings

As discussed in Section 11.4.2, errors in tiling were mitigated using techniques that optimized physical conditions for decreasing the probability of erroneous tile attachments. An alternate approach, introduced in Winfree and Bekbolatov [22], was to design tile sets that were robust to assembly errors by exploiting the mechanism of cooperative bindings. They replaced each tile type by a $k \times k$ block of tile types such that, if an erroneous tile was incorporated in the assembly, there could not be further growth without additional tile mismatches (see Figure 11.7 for an example). Thus, assemblies with incorrect tiles grew much slower and allowed more time for the erroneous attachments to dissociate before the error was locked into place. This *proofreading* could reduce error rates by a square factor over tile sets that did not implement proofreading. However, there is an inherent scale blowup associated with this technique. Reif *et al.* [23] eliminated this scale blowup by giving a compact method to perform proofreading. While these schemes reduce cooperative binding errors, they do not protect against facet nucleation errors and do not scale well with increased k. Chen and Goel [24] introduced the snaked-proofreading technique that guards against both cooperative binding and facet nucleation errors and proved that error rates drop exponentially as a function of k. Both conventional and

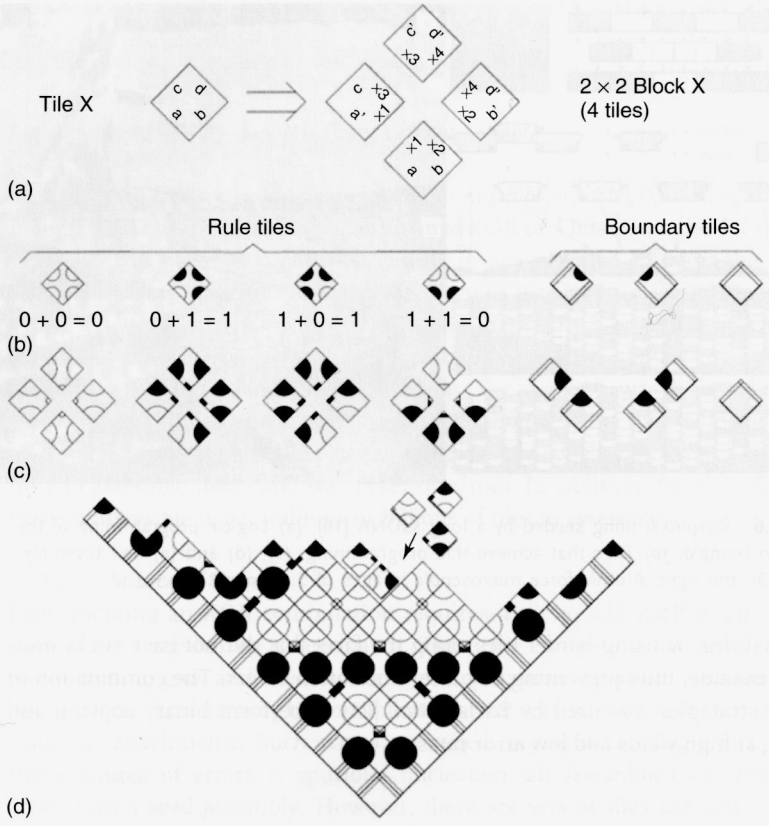

Figure 11.7 Proofreading for Sierpinski tiling [22].

snaked-proofreading systems were experimentally tested out by Chen *et al.* [25], and a 2×2 snaked-proofreading system was shown to reduce facet nucleation errors fourfold as compared to the conventional proofreading technique.

Snaked proofreading reintroduced the problem of scale blowup, but Soloveichik and Winfree [26] fixed this by describing a compact method to implement it. The drawback is that, for certain patterns, this results in an exponential blowup in the descriptional complexity of the pattern (as measured by the number of tile types that assemble it) (Figure 11.8).

Given a tile set, what are the relative concentrations of tile types that minimize errors? Chen and Kao [27] proved that, for rectilinear patterns, carefully setting the concentrations of tile types allows one to achieve minimum errors with the fastest possible assembly time. Instead of working with static tiles, one may imagine dynamic tiles that change state based on some timed signals. Fujibayashi *et al.* [20] and Majumder *et al.* [28] suggested techniques to implement such dynamic tiles and how they might be used to reduce errors in tile assembly.

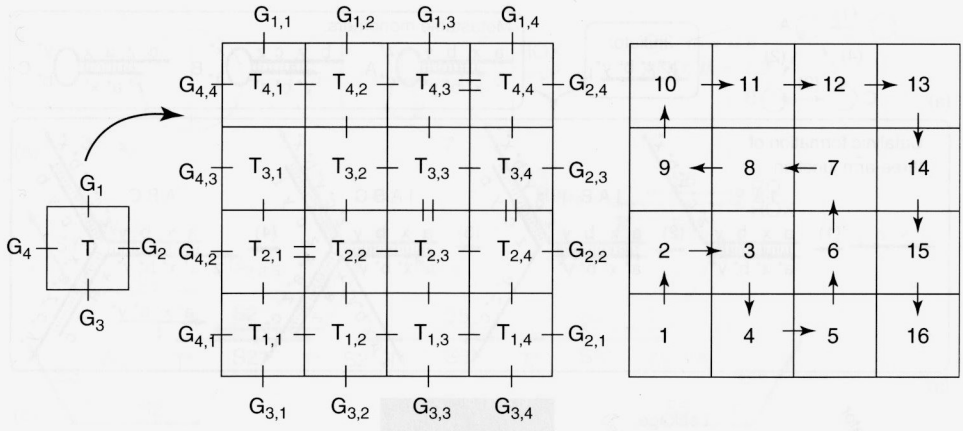

Figure 11.8 Snaked proofreading [24].

11.5
Experimental Advances in Purely Hybridization-Based Computation

Hybridization is the simplest DNA reaction; yet it is powerful enough for performing a wide array of computations. In this section, we will study a few systems built purely on DNA hybridization networks. A DNA nanostructure is formed via the self-assembly of multiple interacting DNA strands. The actual sequence of incorporation of strands into the nanostructure is not always clear. Control over this pathway, in a process termed "*directed self-assembly*", was demonstrated by Yin *et al.* [29]. Their basic unit of construction was the hairpin motif.

Recall that the single-stranded bulge-loop of hairpins undergoes hybridization extremely slowly and thus provides a method for hiding information (making a subsequence unreactive can be thought of as hiding). This information is revealed (activating a subsequence and making it reactive can be thought of as revealing) when the hairpin is opened up via a toehold-mediated strand displacement. The newly revealed subsequence can now reveal other subsequences. Thus one can achieve precise control on the order in which strands get activated and information gets revealed. Yin *et al.* [29] provided an abstract symbolic representational language for programming pathways using the hairpin motifs. Using this language, they programmed and implemented a variety of dynamic functions: catalytic formation of branched junctions, autocatalytic duplex formation via a cross-catalytic circuit, nucleated dendritic growth, and autonomous locomotion of a bipedal walker. Figure 11.9 shows the directed assembly of the catalytic branched junctions.

The notion of hiding and revealing information allows control of the reaction pathway and this idea can be extended to control the computational pathway of a program implemented by DNA molecules. Qian and Winfree [30] built complexes, which they termed "*seesaw*" gates, that allowed them to hide and reveal information along a computational pathway. Figure 11.10 shows how to achieve the AND and OR logic via seesaw reactions.

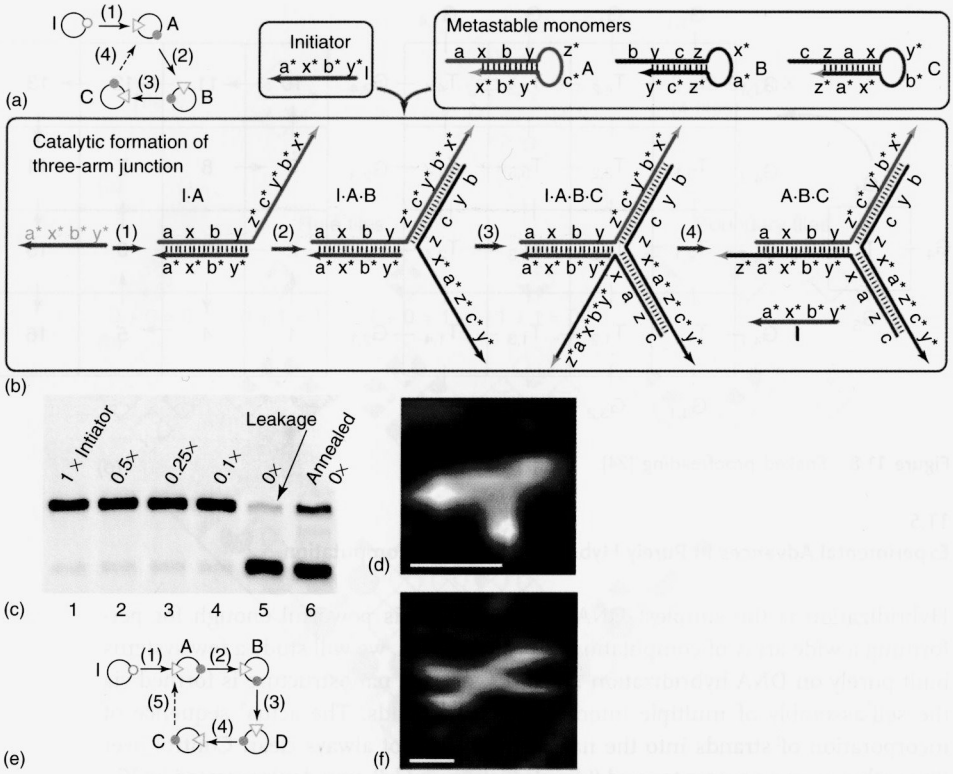

Figure 11.9 (a) Reaction graph for three-arm junctions. (b) Secondary structure mechanism. (c) Agarose gel electrophoresis demonstrating catalytic self-assembly for the three-arm system. (d) AFM image of a three-arm junction. Scale bar: 10 nm. (e) Reaction graph and (f) AFM image for a four-arm junction. Scale bar: 10 nm [29].

The process is represented in an abstract form in Figure 11.10a. Signals x_1 and x_2 are inputs to gate 2 and produce identical outputs that feed into the threshold gate 5. The threshold level *th* determines the logic implemented by this circuit; setting $th = 0.6$ makes the circuit compute the OR of x_1 and x_2, while setting $th = 1.2$ makes the circuit compute the AND of x_1 and x_2. Threshold gates absorb their input signal up to the level specified. Any signal beyond the threshold level gets catalytically amplified to the logical high. Any signal below this threshold is absorbed by the threshold gate and this provides for digital signal restoration in the circuit. Figure 11.10b shows the actual domain-level strand design of the system. The single strands in the system are the signal and fuel strands (that drive the catalytic amplification), while the double-stranded complexes are the various gates. Notice that the gates have two small blue domains T′. These are the toeholds of the gate complexes. In addition, notice how one of the toeholds is revealed while the other is hidden. Seesaw gates essentially work by hiding and revealing these toeholds. Given a circuit consisting of a large number of gates, a set

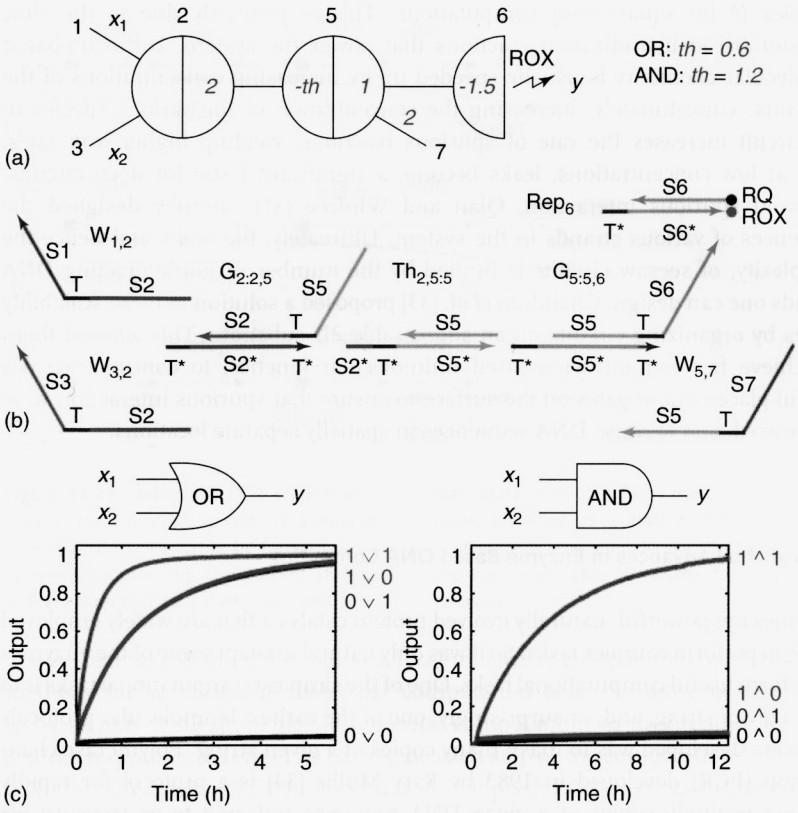

Figure 11.10 Implementing AND and OR logic via seesaw gates. (a): An abstract view of OR and AND seesaw gates. Gate 2 is an integrator, gate 5 is a thresholded catalytic amplifier and gate 6 acts as a downstream drain and releases a fluorescent reporter molecule. (b): The corresponding strand diagram for the gates. (c): Time traces indicating the operation of the OR and AND gates [31].

of input strands will sequentially reveal and hide a specific subset of the toeholds in a specific order, mimicking the process of traversing a computing tree. Qian and Winfree [31] showed how to use the AND and OR gate modules described above to modularly construct large circuits and demonstrated the operation of 4 bit square-root circuits comprised of 130 DNA strands. Furthermore, they showed how to modify the thresholding logic to implement a series of linear threshold gates that can act as a neural network [32]. Their approach allowed them to implement a Hopfield associative memory with four fully connected artificial neurons that, after training *in silico*, remembered four ssDNA patterns and recalled the most similar one when presented with an incomplete pattern.

The scale and complexity of these demonstrations of computing via seesaw gates were truly remarkable. Unfortunately, further scaling of these circuits seems problematic. First, seesaw circuits take a long time (6–10 h) to perform moderately

complex (4 bit square-root) computations. This is primarily due to the slow diffusion-based hybridization reactions that power the system. Diffusion-based bimolecular chemistry is usually speeded up by increasing concentrations of the reactants. Unfortunately, increasing the concentration of the various species in the circuit increases the rate of spurious reactions, yielding higher leak rates. Even at low concentrations, leaks become a significant issue for deep circuits. To avoid spurious interaction, Qian and Winfree [31] carefully designed the sequences of various strands in the system. Ultimately, the scale, and hence the complexity, of seesaw circuits is limited by the number of noninteracting DNA strands one can design. Chandran *et al.* [33] proposed a solution to these scalability issues by organizing circuits on an addressable 2D substrate. This allowed them to achieve fast non-diffusion-based unimolecular kinetics, to control leaks via careful placement of gates on the surface to ensure that spurious interaction were minimized, and to reuse DNA sequences in spatially separate locations.

11.6 Experimental Advances in Enzyme-Based DNA Computing

Enzymes are powerful, naturally evolved protein catalysts that are widely employed by life to perform complex tasks. So it was only natural to adapt some of the enzymes to perform useful computational tasks. One of the simplest computational tasks is to copy a given string, and, unsurprisingly, one of the earliest biomolecular protocols that were developed was to make many copies of a given string. Polymerase chain reaction (PCR) developed in 1983 by Kary Mullis [34] is a protocol for rapidly creating multiple copies of a given DNA sequence (referred to as *template*) via the use of the enzyme polymerase. Polymerase synthesizes a new DNA strand complementary to the DNA template strand by adding free nucleotides in solution that are complementary to the template in the 5′ to 3′ direction. This addition is done to the 3′ end of a primer strand that is hybridized to the template. The newly synthesized strand is then heat-denatured from the template and now both these strands can serve as templates for the creation of their respective complementary strands. Note that in this procedure both the strand and its reverse complements are produced.

The basic PCR technique was adapted for finite-state computations by Sakamoto *et al.* [35] known as *whiplash PCR* (Figure 11.11). The technique uses a strand of DNA that essentially encodes a finite-state machine; the strand is comprised of a sequence of "rule" subsequences each encoding a state transition rule. They are each separated by a stopper sequence that stops the action of DNA polymerase. On each step of the computation, the 3′ end of the DNA strand has a terminal subsequence encoding the state of the computation. A computation step is executed when this 3′ end hybridizes to a portion of a "rule" subsequence, and the action of DNA polymerase extends the 3′ end to a further subsequence encoding a new state. The complex is now thermally denatured and primed to execute the next step of computation.

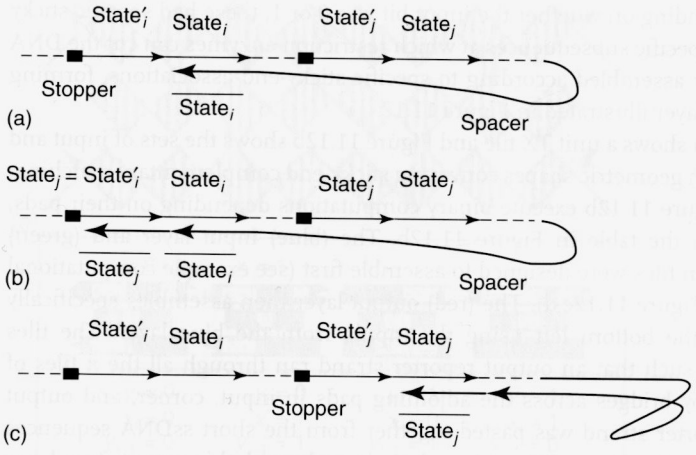

Figure 11.11 Whiplash PCR state transitions. (a) The current state is annealed onto the transition table by forming a hairpin structure. (b) The current state is then extended by polymerase and the next state is copied from the transition table. (c) After denaturation, the new current state is annealed to another part of the transition table to enable the next transition.

Note that whiplash PCR executes a local molecular computation, whereas most methods for autonomous molecular computation (such as those based on the self-assembly of tiles) are global molecular computations as they require multiple distinct molecules that interact to execute each step of the computation.

Neither the original PCR protocol nor the whiplash PCR executes autonomously: they require thermal cycling for each step of their protocols. To overcome this, Walker *et al.* [36] developed isothermal methods for PCR known as "strand displacement amplification" (SDA) using a strand-displacing DNA polymerase. In this method, strands displaced by the DNA polymerase are used for the further stages of the amplification reaction. Reif and Majumder [37] developed an isothermal, autonomously executing version of whiplash PCR that makes use of a strand-displacing polymerase.

The first experimental demonstrations of computation via DNA tile assembly were done in 2000 by Mao *et al.* [12] with the help of the ligase enzyme. Ligases repair single-stranded discontinuities in dsDNA molecules. Mao *et al.* [12] demonstrated a two-layer linear assembly of TX tiles that executed bit-wise cumulative XOR computation. This computation takes as input n bits and computes n output bits, where the ith output bit is the XOR of the first i input bits. This computation frequently occurs when one determines the output bits of a full-carry binary adder circuit found in most microprocessors.

These experiments provided initial answers to some of the most basic questions of how autonomous molecular computation might be done: How can one provide data input to a molecular computation using DNA tiles? In this experiment, the input sequence of n bits was defined using a specific series of "input" tiles, with the input bits (1's and 0's) encoded by distinct short subsequences. Two different

tile types (depending on whether the input bit was 0 or 1, these had specific sticky ends and also specific subsequences at which restriction enzymes can cut the DNA backbone) were assembled according to specific sticky-end associations, forming the blue input layer illustrated in Figure 11.12.

Figure 11.12a shows a unit TX tile and Figure 11.12b shows the sets of input and output tiles with geometric shapes conveying sticky-end complementary matching. The tiles of Figure 11.12b execute binary computations depending on their pads, as indicated by the table in Figure 11.12b. The (blue) input layer and (green) corner condition tiles were designed to assemble first (see example computational assemblies in Figure 11.12c,d). The (red) output layer then assembles specifically starting from the bottom left using the inputs from the blue layer. The tiles were designed such that an output reporter strand ran through all the n tiles of the assembly by bridges across the adjoining pads in input, corner, and output tiles. This reporter strand was pasted together from the short ssDNA sequences within the tiles using ligase. When the solution was heated, this output strand was isolated and identified. The output data was read by experimentally determining the sequence of cut sites (see below). In principle, the output could be used for subsequent computations.

The next question of concern is: How can one execute a step of computation using DNA tiles? To execute steps of computation, the TX tiles were designed to have pads at one end that encoded the cumulative XOR value. In addition, since the reporter strand segments ran through each such tile, the appropriate input bit was also provided within its structure. These two values implied that the opposing pad on the other side of the tile would be the XOR of these two bits.

A final question is: How can one determine and/or display the output values of a DNA tiling computation? The output in this case was read by determining which of the two possible cut sites (endonuclease cleavage sites) was present at each position in the tile assembly. This was executed by first isolating the reporter strand, and then digesting separate aliquots with each endonuclease separately and the two together. Finally, these samples were examined by gel electrophoresis and the output values were displayed as banding patterns on the gel. Another method for output is the use of atomic force microscopy (AFM) observable patterning. The patterning was made by designing the tiles computing a bit 1 to have a stem loop protruding from the top of the tile. This molecular patterning was clearly observable under appropriate AFM imaging conditions.

An alternative method for autonomous execution of a sequence of finite-state transitions was subsequently developed by Shapiro and Benenson [38, 53]. Their technique essentially operated in the reverse of the assembly methods described above, and can be thought of as disassembly. They began with a linear dsDNA nanostructure whose sequence encoded the inputs, and then they executed a series of steps that digested the DNA nanostructure from one end (Figure 11.13).

On each step, a sticky end at one end of the nanostructure encoded the current state, and the finite transition was determined by hybridization of the current sticky end with a small "rule" nanostructure encoding the finite-state transition rule. Then a restriction enzyme, which recognized the sequence encoding the current input as

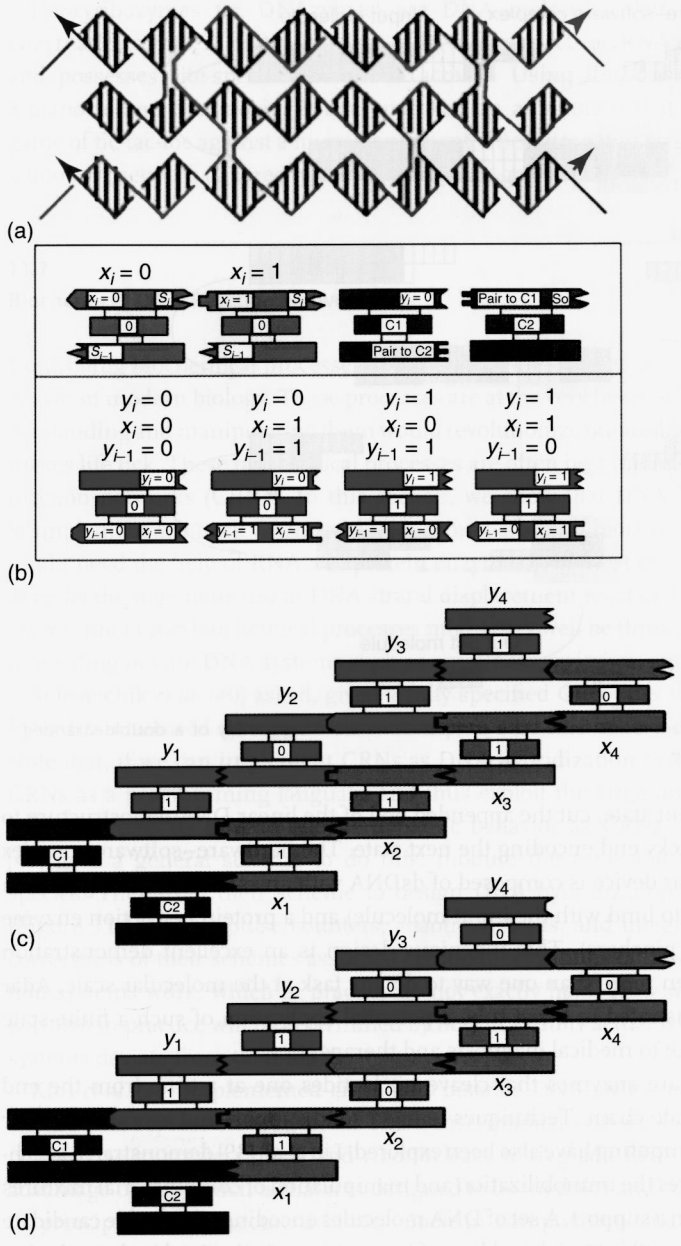

Figure 11.12 Sequential Boolean computation via a linear DNA tiling assembly. (a) TX tile used in assembly. (b) Set of TX tiles providing logical programing for computation. (c,d) Example of resulting computational tilings. (Reprinted by permission from Macmillan Publishers Ltd: Nature Mao, C. *et al.*, Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, 407, 49–496. © 2000.)

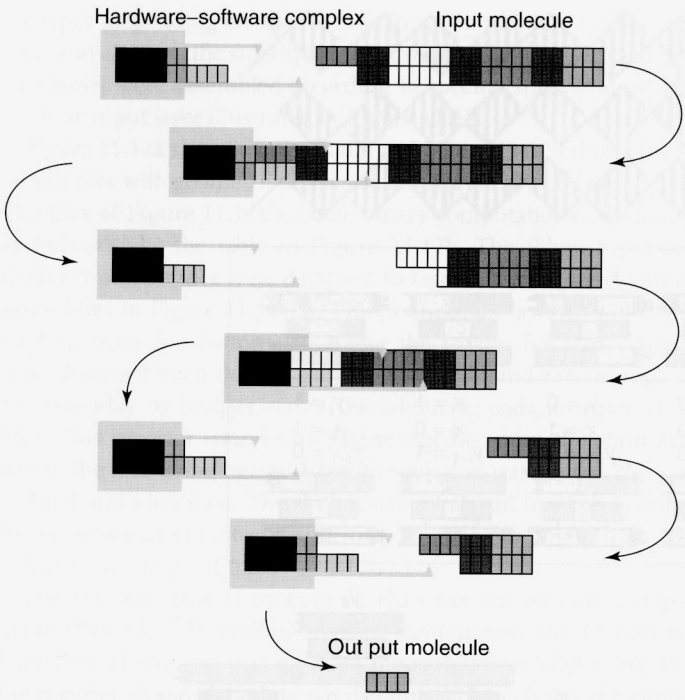

Figure 11.13 Autonomous finite-state computations via disassembly of a double-stranded DNA nanostructure.

well as the current state, cut the appended end of the linear DNA nanostructure to expose a new sticky end encoding the next state. The hardware–software complex for this molecular device is composed of dsDNA with an ssDNA overhang (shown at top left ready to bind with the input molecule) and a protein restriction enzyme (shown as gray pinchers). This ingenious design is an excellent demonstration that there is often more than one way to do any task at the molecular scale. Adar *et al.* [39] demonstrated in a test tube a potential application of such a finite-state computing device to medical diagnosis and therapeutics.

Exonucleases are enzymes that cleave nucleotides one at a time from the end of a polynucleotide chain. Techniques using a solid support and exonucleases for biomolecular computing have also been explored. Liu *et al.* [39] demonstrated a technique that involves the immobilization and manipulation of combinatorial mixtures of DNA stands on a support. A set of DNA molecules encoding all possible candidate solutions to the combinatorial problem of interest was synthesized and attached to the surface. Successive rounds of hybridization operations and exonuclease digestion were employed to identify and eliminate those strands that were not solutions to the problem. Upon completion of all the rounds, the solution to the problem was identified using PCR to amplify the remaining molecules. This method was used to solve an NP-complete problem, namely a four-variable 3SAT problem.

Deoxyribozymes (or DNAzymes) are DNA molecules that possess catalytic enzymatic activity. A particular class of DNAzymes acts as RNA nicking enzymes and possesses site-specific restriction activity. Using this class of DNAzymes, Stojanovic and Stefanovic [2] demonstrated an automaton that played a perfect game of tic-tac-toe against a human opponent. The automaton was implemented as a Boolean network of three types of DNAzymes gates: YES, NOT, and AND gates.

11.7 Biochemical DNA Reaction Networks

Controlling biochemical processes at the cellular and subcellular levels is a key endeavor of modern biology. These processes are at the very heart of life itself, and understanding and manipulating them would revolutionize our understanding of what makes life tick. These biochemical processes are often best understood as chemical reaction networks (CRNs). In this section, we show that DNA has the potential to implement arbitrary CRNs, and thus mimic arbitrary biochemical processes. It might need the help of RNA and protein enzymes, but most of the heavy lifting is done by the ingenious use of DNA strand displacement reactions. The path to programming *in vivo* biochemical processes might very well be through implementing interesting *in vitro* DNA systems such as amplifiers, switches, and oscillators.

Soloveichik *et al.* [40] asked, given a fully specified CRN does there exist a DNA hybridization-based system that can mimic, *in vitro*, the behavior of this network. Note that, if we can implement CRNs as DNA hybridization systems, we can use CRNs as a programming language and thus exploit the large preexisting body of work that tells us how to encode dynamic behavior as CRNs. Soloveichik *et al.* [40] gave a general scheme to do this: modulo the time and concentration of species. They used their scheme to design DNA hybridization-based oscillators (Figure 11.14), 2 bit pulse counters, chaotic systems, and integral counters. The correctness of their schemes assumed certain idealizations of how DNA hybridization systems work, which in practice do not exactly hold. How well their systems perform in practice will be determined by how gracefully actual DNA hybridization systems deviate from their ideal.

Kim *et al.* [41] implemented an *in vitro* bistable DNA switch regulated by RNA signals that repress transcription. The transcriptional system was fueled by RNA polymerase to create signals and ribonuclease to degrade RNA signals. The key advantage of this switch is that it is designed to be modular and composable. Thus, in principle it can execute arbitrary Boolean computation and also implement neural networks [42]. In fact, this same switch was used by Kim and Winfree [43] to construct three kinds of oscillators: a two-switch negative feedback oscillator, an amplified version of a negative feedback oscillator, and a three-switch ring oscillator. Franco *et al.* [44] have recently shown (Figure 11.15) that the two-switch negative feedback oscillator can drive a load. They used it to open and close a DNA tweezer and also released a functional RNA molecule.

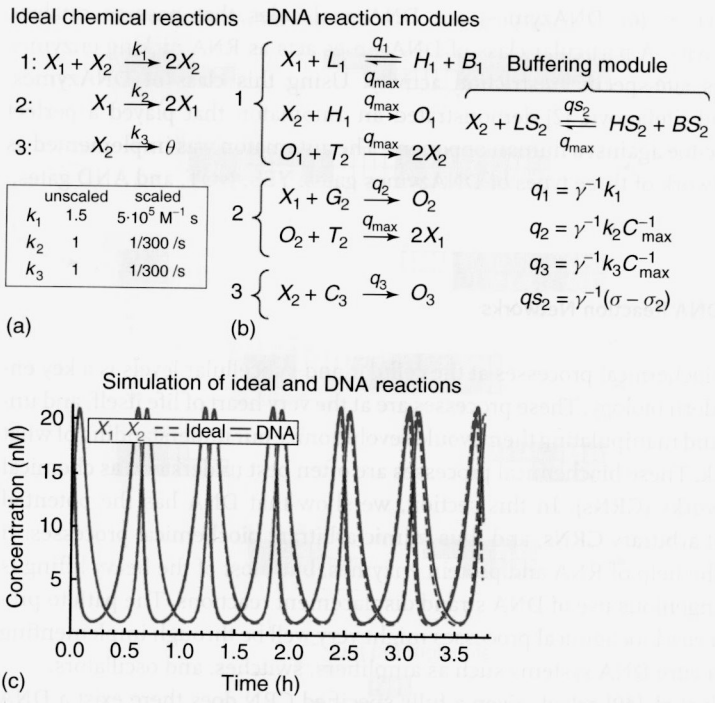

Figure 11.14 Lotka–Volterra chemical oscillator CRN using DNA [40].

11.8 Conclusion: Challenges in DNA-Based Biomolecular Computation

We have discussed many methods for DNA-based biomolecular computation, including some very impressive experimental demonstrations, but there are many challenges still remaining.

11.8.1 Scalability of Biomolecular Computations

Scalability of biomolecular computations are currently limited by a number of highly interconnected issues. We briefly discuss them below.

- The rate of errors: We have discussed approaches for reducing various types of errors in biomolecular computations including leakages of logical gates in the case of hybridization reaction circuits and also errors in the case of algorithmic self-assembly of DNA tiles. While some of these methods have been implemented experimentally, most of them remain as experimental challenges. These error-correction methods need to be improved to make them experimentally implementable and effect further improvements in the complexity of biomolecular computations.

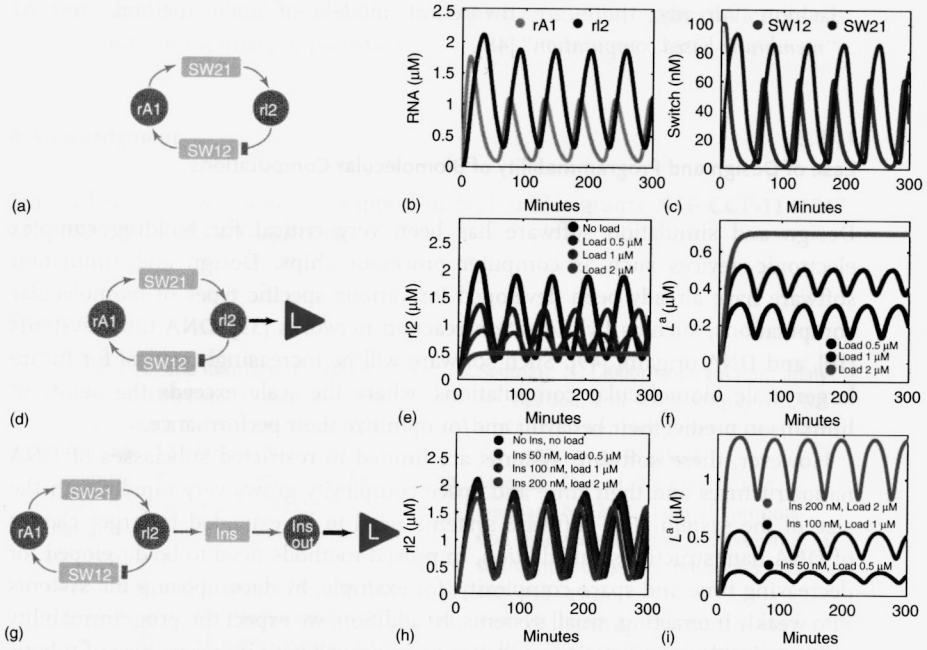

Figure 11.15 Two-switch negative feedback oscillator driving a load. Caption adapted from [44] (a): Diagram for the simple model for the oscillator. (b): Time traces for the oscillator species rA1 and rI2. (c): Time traces for the oscillator species SW12 and SW21. (d): Oscillator scheme with consumptive load coupled to rI2. (e, f): Time traces for the oscillator and load for consumptive coupling on rI2. (g): Oscillator scheme with consumptive insulating circuit and consumptive load. (h, i): Time traces for the oscillator and load when the insulating genelet is used to amplify rI2.

- The speed of biomolecular computation: The speed of many biomolecular computations is limited principally by diffusion delay times – the time for distinct molecules to find each other so as to initiate a reaction. These delays are substantially increased by the number of molecules that need to find each other to initiate a reaction, and hence bimolecular reactions are generally preferred to reactions involving more than two molecules at a time. The diffusion delay can be decreased by increasing the concentration of the reactants, but this can increase the rate of errors as well. One promising approach to improve the rate of biomolecular computation that we have discussed is to make the reactions local, for example, tethering the reacting molecules so they stay in the same relative vicinity.
- The modularity of biomolecular computations: Currently, biomolecular computations are only weakly modular, and there is considerable cross-talk between distinct reactions in a test tube. The modularity may be increased by the use of techniques such as artificial liposomes (45, 46) or DNA nanostructure boxes [47], which segregate the DNA material and their reactions. These may need to be designed to merge and split in a dynamic and/or programmable

fashion. Already, there are theoretical models of such methods termed *"membrane-based computation"* [48].

11.8.2 Ease of Design and Programmability of Biomolecular Computations

Design and simulation software has been very critical for building complex electronic devices such as computer processor chips. Design and simulation software have already been developed for various specific types of biomolecular computations, such as hybridization reaction networks [31], DNA tiling systems [10], and DNA origami [49]. Such software will be increasingly critical for future larger scale biomolecular computations, where the scale exceeds the ability of humans to predict their behavior and/or optimize their performance.

However, these software systems are limited to restricted subclasses of DNA nanostructures and their time and space complexity grows very rapidly with the size of the system. The software systems need to be extended to larger classes of DNA nanostructures. In addition, improved methods need to be developed for decreasing time and space complexity, for example, by decomposing the systems into weakly interacting, small systems. In addition, we expect the programmability of biomolecular computations will also be improved with increased use of robotic manipulation of reagents (see [50]) and microfluidics technologies [51].

11.8.3 *In Vivo* Biomolecular Computations

Some of the most potential promising applications of biomolecular computations are *in vivo* applications. These may include the idea of a DNA doctor [38] that makes use of a biomolecular device within a cell that monitors the cells and diagnoses diseases determined by conditions such as underexpression or overexpression of some of the cell's messenger RNA and responds by the release of nucleic acids that mitigate the disease. However, only a very few biomolecular computations [52] have been demonstrated to have *in vivo* viability. It is an open challenge to develop robust techniques for executing biomolecular computations within a living cell. One recent promising technique is the use of liposomes within the cell to protect synthetic DNA devices from digestion by the cell.

11.8.4 Conclusions

In spite of the many considerable challenges enumerated in this section, there has been very impressive scalability of experimental demonstrations of biomolecular computations, riveling the rate of improvements in VLSI technologies and computer architectures in the 1970s. The community of scientists working on biomolecular computations has also to likewise considerably increase, both in

numbers and diversity of their disciplines, to overcome the many challenges that will need interdisciplinary approaches.

Acknowledgment

The authors wish to thank the support of NSF under grants **NSF CCF-1141847**, **CCF-0829797**, and **CCF-0829798**.

References

1. Zhang, D.Y. and Winfree, E. (2009) Control of DNA strand displacement kinetics using toehold exchange. *J. Am. Chem. Soc.*, **131** (48), 17303–17314.
2. Stojanovic, M.N. and Stefanovic, D. (2003) A deoxyribozyme-based molecular automaton. *Nat. Biotechnol.*, doi: 10.1038/nbt862
3. Garey, M.R. and Johnson, D.S. (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Company.
4. Feynman, R. (1959) There's Plenty of Room at the Bottom.
5. Adleman, L. (1994) Molecular computation of solutions to combinatorial problems. *Science*, **266**, 1021–1024.
6. Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., and Adleman, L.M. (1996) A sticker based model for DNA computation, *Proceedings of the Second Annual Meeting on DNA Based Computers* , American Mathematical Society, 1–29.
7. Lipton, R. (1995) DNA solution of hard computational problems. *Science*, **268**, 542–545.
8. Reif, J.H. (2002) DNA computation – perspectives: successes and challenges. *Science*, **296**, 478–479.
9. Winfree, E. (1998b) Algorithmic self-assembly of DNA. PhD Thesis. Caltech.
10. Winfree, E., Liu, F., Wenzler, L.A., and Seeman, N.C. (1998) Design and self-assembly of two-dimensional DNA crystals. *Nature*, **394** (6693), 529–544.
11. LaBean, T., Yan, H., Kopatsch, J., Liu, F., Winfree, E., Reif, J., and Seeman, N. (2000) Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. *J. Am. Chem. Soc.*, **122** (9), 1848–1860.
12. Mao, C., Labean, T., Reif, J., and Seeman, N. (2000) Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, **407**, 493–496.
13. Yan, H., Feng, L., LaBean, T., and Reif, J. (2003a). Parallel molecular computation of pair-wise XOR using DNA string tile. *J. Am. Chem. Soc.*, **12**.
14. Yan, H., Feng, L., LaBean, T., and Reif, J. (2003) Parallel molecular computations of pairwise exclusive-or (XOR) using DNA string tile self-assembly, *Nature*, **125** (47), 14246–14247.
15. Rothemund, P.W.K. (2000) Using lateral capillary forces to compute by self-assembly, *Proc. Natl. Acak. Sci.*, **97** (3), 984–989, doi: 10.1073/pnas.97.3.984
16. Rothemund, P.W.K., Papadakis, N., and Winfree, E. (2004) Algorithmic self-assembly of DNA sierpinski triangles, *PLoS Biol.*, **2** (12), 424–436.
17. Winfree, E. (1998a) Simulations of Computing by Self-Assembly. Technical report. California Institute of Technology, CaltechCSTR:1998.22Persistent *http://resolver.caltech.edu/CaltechCSTR:1998.22.*
18. Schulman, R., Lee, S., Papadakis, N., and Winfree, E. (2004) in *DNA Computing*, Vol. **9** (eds J Chen and J Reif), Springer-Verlag, Berlin, pp. 108–125.
19. Barish, R.D., Rothemund, P.W.K., and Winfree, E. (2005) *Nano Lett.*, **5**, 2586–2259.

20. Fujibayashi, K., Zhang, D.Y., Winfree, E., and Murata, S. (2009) Error suppression mechanisms for DNA tile self-assembly and their simulation. *Nat. Comput.*, **8** (3), 589–612.
21. Barish, R.D. *et al.* (2009) An information-bearing seed for nucleating algorithmic self-assembly, *Proc. Natl. Acak. Sci.*, **106** (15), 6054–6059.
22. Winfree, E. and Bekbolatov, R. (2003) Proofreading tile sets: error correction for algorithmic self-assembly. *DNA Comput.*, LNCS, **2943**, 126–144.
23. Reif, J., Sahu, S., and Yin, P. (2004) Compact error-resilient computational DNA tiling assemblies. *DNA Comput.*, **3384**, 293–307.
24. Chen, H.L. and Goel, A. (2005) Error free self-assembly using error prone tiles, *DNA Computing*, **3384**, 702–707.
25. Chen, H.L., Schulman, R., Goel, A., and Winfree, E. (2007) Reducing facet nucleation during algorithmic self-assembly, *Nano Letters* **7**, 2913–2919.
26. Soloveichik, D. and Winfree, E.. (2006) Complexity of compact proofreading for self-assembled patterns. *Proceedings of DNA Computing 11*, Springer-Verlag Berlin Heidelberg, LNCS **3892**, 305–324.
27. Chen., H.L. and Kao., M.Y. (2011) *DNA Computing and Molecular Programming*, Lecture Notes in Computer Science, Vol. **6518**, Springer, Berlin, Heidelberg, p. 13, ISBN: 978-3-642-18304-1.
28. Majumder, U. *et al.* LaBean, T.H., and Reif, J. (2008) *Activatable Tiles for Compact, Robust Programmable Assembly and other Applications, DNA 13*, Springer-Verlag, LNCS 4848, 15–25, Newyork.
29. Yin, P., Choi, H.M., Calvert, C.R., and Pierce, N.A. (2008) Programming biomolecular self-assembly pathways, *Nature*, **451** (7176), 318–322.
30. Qian, L. and Winfree, E. (2011) A simple DNA gate motif for synthesizing large-scale circuits, *J. R. Soci. Interface*, *http://dx.doi.org/10.1098/rsif.2010.0729*, doi: 10.1098/rsif.2010.0729
31. Qian, L. and Winfree, E. (2011) Scaling up digital circuit computation with DNA strand displacement cascades, *Science*, **332** (6034), 1196–1201.
32. Qian, L., Winfree, E., and Bruck, J. (2011) Neural network computation with DNA strand displacement cascades. *Nature*, **475**, 368–372.
33. Chandran, H., Gopalkrishnan, N., Phillips, A., and Reif, J. (2011) Localized hybridization circuits, in *DNA17*, Springer-Verlag Berlin, Heidelberg, 64–83.
34. Mullis, K., Faloona, F., Scharf, S., Saiki, R., Horn, G., and Erlich, H. (1986) Specific enzymatic amplification of DNA in vitro: the polymerase chain reaction. *Cold Spring Harb. Symp. Quant. Biol.*, **51**, 263.
35. Sakamoto, K., Kiga, D., Momiya, K., Gouzu, H., Yokoyama, S., Ikeda, S., Sugiyama, H., and Hagiya, M. (1998) State transitions with molecules. Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16–19, 1998.
36. Terrance Walker, G., Fraiser, M.S., Schram, J.L., Little, M.C., Nadeau, J.G., and Malinowski, D.P. (1992) Strand displacement amplification---an isothermal, *in vitro* DNA amplification technique. *Nucleic Acids Res.*, **20** (7), 1691–1696.
37. Reif, J.H. and Majumder, U. (2009) Isothermal Reactivating Whiplash PCR for Locally Programmable Molecular Computation, *DNA Computing, Lecture Notes in Computer Science* Vol. 5347 (eds A. Goel, F. Simmel, P. Sosík, Springer Berlin, Heidelberg, pp. 41–56, ISBN: 978-3-642-03075-8, *http://dx.doi.org/10.1007/978-3-642-03076-5_5*.
38. Shapiro, E. and Benenson, Y. (2006) Bringing DNA computers to life. *Sci. Am.*, **294**, 44–51.
39. Liu, Q., Wang, L., Frutos, A.G., Condon, A.E., Corn, R.M., and Smith, L.M. (2000) DNA computing on surfaces. *Nature*, **403**, 175–179.
40. Soloveichik, D., Seelig, G., and Winfree, E. (2010) DNA as a universal substrate for chemical kinetics.
41. Kim, J., White, K.S., and Winfree, E. (2006) Construction of an in vitro bistable circuit from synthetic transcriptional switches, **2** (Art. no: 68), *Mol. Syst. Biol.*, doi: 10.1038/msb4100099

42. Kim, J., Hopfield, J.J., and Winfree, E. (2004) Neural network computation by in vitro transcriptional circuits. *Adv. Neural Inf. Process. Syst. (NIPS)*, **17**, 681–688.
43. Kim, J. and Winfree, E. (2011) Synthetic in vitro transcriptional oscillators.
44. Păun, G. (1998) Computing with Membranes. *J. Comput. Syst. Sci.*, **61**, 108–143.
45. Hamada, T., Sugimoto, R., Vestergaard, M.C., Nagasaki, T., and Takagi, M. (2010) Membrane disk and sphere: controllable mesoscopic structures for the capture and release of a targeted object. *J. Am. Chem. Soc.*, **132** (30), 10528–10532.
46. Thompson, M.P., Chien, M.P., Ku, T.H., Rush, A.M., and Gianneschi, N.C. (2010) Smart lipids for programmable nanomaterials. *Nano Lett.*, **10** (7), 2690–2693.
47. Andersen, E.S., Dong, M., Nielsen, M.M., Jahn, K., Subramani, R., Mamdouh, W., Golas, M.M., Sander, B., Stark, H., Oliveira, C.L.P., Pedersen, J.S., Birkedal, V., Besenbacher, F., Gothelf, K.V., and Kjems, J. (2009) Self-assembly of a nanoscale DNA box with a controllable lid. *Nature*, **459**, 73–76.
48. Păun, G. (1998) Computing with Membranes. TUCS Report 208, Turku Center for Computer Science.
49. Rothemund, P.W.K. (2006) Folding DNA to create nanoscale shapes and patterns. *Nature*, **440**, 297–302.
50. Shapiro, E. and Robolab (2009) A Robot Programming Language.
51. Kirby, B.J. (2010) *Micro- and Nanoscale Fluid Mechanics: Transport in Microfluidic Devices*, Cambridge University Press.
52. Modi, S., Swetha, M.G., Goswami, D., Gupta, G.D., Mayor, S., and Krishnan, Y. (2009) A DNA nanomachine maps spatiotemporal pH changes in living cells. *Nat. Nanotechnol.*, **4**, 325–330.
53. Benenson, Y., Gil, B., Ben-Dor, U., Adar, R., and Shapiro, E. (2004) An autonomous molecular computer for logical control of gene expression. *Nature*, **429** (6990), 423–429.
54. Winfree, E., Yang, X., and Seeman, N. (1996) Universal computation via self-assembly of DNA: some theory and experiments. *DNA Based Computers II*, DIMACS, **44**, 191–213.