31

DNA Computing: A Research Snapshot

Lila Kari
University of Western Ontario

Kalpana Mahalingam
Indian Institute of Technology

31.1 Introduction

During the last decades, we have witnessed exciting new developments in computation theory and practice, from entirely novel perspectives. DNA Computing, known also under the names of molecular computing, biocomputing, or biomolecular computing, is an emergent field lying at the crossroads of computer science and molecular biology. It is based on the idea that data can be encoded as DNA strands, and molecular biology tools can be used to perform arithmetic and logic operations.

This chapter intends to give the reader a basic understanding of the tools and methods used in DNA computing, and a snapshot of theoretical and experimental research in this field. It includes descriptions of a representative DNA computing experiment, of the design of autonomous and programmable molecular computers, and of models of DNA memories. It describes research into computation in and by living cells, as well as into DNA self-assembly, a process that can be used for computation or can produce either static DNA nanostructures or dynamic DNA nanomachines. This chapter is intended to offer the reader a glimpse of the astonishing world of DNA computing, rather than being an exhaustive review of the research in the field.

The chapter is organized as follows.* Section 31.2 introduces basic molecular biology notions about DNA as an information-encoding medium, and the bio-operations that can be performed on DNA strands. As a representative example of a DNA bio-algorithm, in which a computational problem is solved exclusively by bio-operations performed on DNA strands in test tubes, Section 31.3 describes an experiment demonstrated by Adleman's team, that solved a 20-variable instance of the 3-SAT problem. Section 31.4 exemplifies the research on molecular autonomous programmable DNA computers by the description of Benenson automata and their potential applications to smart drug design. Section 31.5 presents research into DNA memory technology such as nested primer molecular memory (Section 31.5.1), and organic DNA memory (Section 31.5.2), as well as explores the topic of optimal DNA sequence design for DNA computing purposes (Section 31.5.3). Section 31.6 explores the fascinating possibilities offered by *in vivo* computing, that is, computation in and by living cells, as well as the potential insights it could offer into understanding the computational processes in nature. Section 31.7 describes one of the most important directions of research in DNA computing, with significant implications for the rapidly evolving field of nanotechnology: DNA computing by self-assembly. It includes descriptions of experimental computation by self-assembly (Section 31.7.1), of static intricate DNA nanostructures (Section 31.7.2), and of impressive DNA nanomachines (Section 31.7.3). Finally, Section 31.8 offers conclusions and brief comments on the latest developments in the field of DNA computing.

31.2 Molecular Biology Basics

This chapter was written with the expectation that the reader is a computer scientist with a limited knowledge of biology. This section provides the basic notions of molecular biology (DNA structure and DNA bio-operations) necessary for understanding the text.

DNA (*deoxyribonucleic acid*) is found in every cellular organism as the storage medium for genetic information, [39]. DNA is a polymer whose monomer units are *nucleotides*, and the polymer is known as a "polynucleotide." More precisely, DNA is a linear chain made up of four different types of nucleotides, each consisting of a base (Adenine, Cytosine, Guanine, or Thymine) and a sugar-phosphate unit. The sugar-phosphate units are linked together by covalent bonds to form the backbone of the DNA single strand. Since nucleotides may differ only by their bases, a DNA strand can be viewed as simply a word over the four-letter alphabet {A, C, G, T}. A DNA single strand has an orientation, with one end known as the $5'$ end, and the other as the $3'$ end, based on their chemical properties. By convention, a word over the DNA alphabet represents the corresponding DNA single strand in the $5' \longrightarrow 3'$ orientation, that is, GGTTTTT stands for the DNA single strand $5'$-GGTTTTT-$3'$. A short single-stranded polynucleotide chain, usually less than 20 nucleotides long, is called an *oligonucleotide*. A DNA strand is sometimes called an *n-mer*, where *n* signifies its length, that is, the number of its nucleotide monomers.

A crucial feature of DNA single strands is their *Watson–Crick complementarity*, [91]: A is complementary to T, and G is complementary to C. Two complementary DNA single strands with opposite orientation will bind to each other by hydrogen bonds between their individual bases to form a stable DNA double strand with the backbones at the outside and the bound pairs of bases lying inside. In a DNA double strand, two complementary bases situated on opposite strands, and bound together by hydrogen bonds, form a *base-pair (bp)*.

* The general audience for whom this paper is intended, our fields of expertize, and the space available for this chapter, affected both the depth and the breadth of our exposition. In particular, the list of research topics presented here is by no means exhaustive, and it is only meant to give a sample of DNA computing research. Moreover, the space devoted to various fields and topics was influenced by several factors and, as such, has no relation to the respective importance of the field or the relative size of the body of research in that field.

As an example, the DNA single strand 5′- AAAAACC - 3′ will bind to the DNA single strand 5′-GGTTTTT-3′ to form the 7 base-pair-long (7bp) double strand

$$\begin{array}{c} 5' - \text{AAAAACC} - 3' \\ 3' - \text{TTTTTGG} - 5' \end{array}.$$

Formally, the Watson–Crick complement of a DNA single strand w will be denoted by $\text{WK}(w)$ or $\overleftarrow{w}$. Using our convention on directionality of strands, note that $\text{WK}(\text{AAAAACC}) = \text{GGTTTTT}$.

Another nucleic acid that has been used for computations is RNA, [23]. *Ribonucleic acid* or *RNA* is a nucleic acid that is similar to DNA, but differs from it in three main aspects: RNA is typically single-stranded while DNA is usually double-stranded, RNA nucleotides contain the sugar ribose, while DNA nucleotides contain the sugar deoxyribose, and in RNA the nucleotide Uracil, U, substitutes for Thymine, which is present in DNA.

The genome consists of DNA sequences, some of which are genes that can be transcribed into *messenger RNA (mRNA)*, and then translated into proteins according to the *genetic code* that maps each 3-letter RNA segment (called *codon*) into an amino acid. Several designated triplets, called start (stop) codons, signal the initiation (termination) of a translation. A protein is a sequence over the 20-letter alphabet of amino acids.

There are many possible *DNA bio-operations* that can be used for computations, [2,38,64], such as: hybridization by Watson–Crick complementarity, cut-and-paste operations achievable by enzymes, the synthesis of desired DNA strands up to a certain length, making exponentially many copies of a DNA strand, the separation of strands by length, the extraction of DNA strands that contain a certain subsequence, and reading-out a DNA strand. These bio-operations, some briefly explained later, have all been used to control DNA computations and DNA robotic operations.

DNA single strands with opposite orientation will join together to form a double helix in a process called *base-pairing*, *annealing*, or *hybridization*. The reverse process—a double-stranded helix coming apart to yield its two constituent single strands—is called *melting*. As the name suggests, melting is achieved by raising the temperature, and annealing by lowering it.

One class of enzymes, called *restriction endonucleases*, each recognize a specific short sequence of DNA, known as a *restriction site*. Any double-stranded DNA that contains the restriction site within its sequence is cut by the enzyme at that location in a specific pattern. Depending on the enzyme, the cutting operation leaves either two "blunt-ended" DNA double strands or, more often, two DNA strands that are double-stranded but have single-stranded overhangs known as "sticky-ends." Another enzyme, called *DNA ligase*, can link together two partially double-stranded DNA strands with complementary sticky-ends, by sealing their backbones with covalent bonds. The process is called *ligation*.

The separation of DNA strands by length is possible by using a technique called *gel electrophoresis*. The negatively charged DNA molecules are placed at the top of a wet gel, to which an electric field is applied, drawing them to the bottom. Larger molecules travel more slowly through the gel. After a period, the molecules spread out into distinct bands according to their size.

The extraction of DNA single strands that contain a specific subsequence v, from a heterogeneous solution of DNA single strands, can be accomplished by *affinity purification*. After synthesizing strands Watson–Crick complementary to v, and attaching them to magnetic beads, the heterogeneous solution is passed over the beads. Those strands containing v anneal to its Watson–Crick complementary sequence and are retained. Strands not containing v pass through without being retained.

The *DNA polymerase* enzymes perform several functions including the replication of DNA by a process called *Polymerase Chain Reaction*, or *PCR*. The PCR replication reaction requires a guiding DNA single strand called *template*, and an oligonucleotide called *primer*, that is annealed to the template. The DNA polymerase enzyme then catalyzes DNA synthesis by successively adding

nucleotides to one end of the primer. The primer is thus extended at its $3'$ end, in the direction $5'$ to $3'$ only, until the desired strand is obtained that starts with the primer and is complementary to the template. If two primers are used, the result is the exponential multiplication of the subsequence of the template strand that is flanked by the two primers, in a process called *amplification*, schematically explained in the following. For the purpose of this explanation, if x is a string of letters over the DNA alphabet {A, C, G, T}, then $\overline{x}$ will denote its simple complement, for example,

$$\overline{\text{AACCTTGG}} = \text{TTGGAACC}.$$

Let us assume now that one desires to amplify the subsequence between x and y from the DNA double strand $\begin{matrix} 5'-\alpha x\beta y\delta-3' \\ 3'-\bar{\alpha}\bar{x}\bar{\beta}\bar{y}\bar{\delta}-5' \end{matrix}$, where $\alpha, x, \beta, y, \delta$ are DNA segments. Then, one uses as primers the strand x and the Watson–Crick complement of y. After heating the solution and thus melting the double-stranded DNA into its two constituent strands, the solution is cooled and the Watson–Crick complement of y anneals to the "top" strand, while x anneals to the "bottom" strand. The polymerase enzyme extends the $3'$ ends of both primers into the $5'$ to $3'$ direction, producing partially double-stranded strands $\begin{matrix} 5'-\alpha x\beta y\delta-3' \\ 3'-\bar{\alpha}\bar{x}\bar{\beta}\bar{y}\ \ -5' \end{matrix}$ and $\begin{matrix} 5'-\ \ x\beta y\delta-3' \\ 3'-\bar{\alpha}\bar{x}\bar{\beta}\bar{y}\bar{\delta}-5' \end{matrix}$. In a similar fashion, the next heating–cooling cycle will result in the production of the additional strands $5'-x\beta y-3'$ and $3'-\bar{x}\bar{\beta}\bar{y}-5'$. These strands are Watson–Crick complementary and will, from now on, be produced in excess of the other strands, since both are replicated during each cycle. At the end, an order of 2^n copies of the desired subsequences flanked by x and y will be present in the solution, where n is the number of the heating–cooling cycles.

The earlier and other bio-operations have all been harnessed in biocomputing. To encode information using DNA, one can choose an encoding scheme mapping the original alphabet onto strings over {A, C, G, T}, and proceed to synthesize the information-encoding strings as DNA single strands. A biocomputation consists of a succession of bio-operations, [20], such as the ones described in this section. The DNA strands representing the output of the biocomputation can then be read out (using a sequencer) and decoded.

31.3 Adleman's 3-SAT Experiment

The idea of using DNA molecules as the working elements of a computer goes back to 1994, [1], when Adleman solved a 7-node instance of the Hamiltonian Path Problem by exclusively using DNA strands to encode information, and molecular biology techniques as algorithmic instructions. This section describes another, more complex, DNA computing experiment that solves a large instance of the 3-SAT problem, [15].

The 3-SAT problem is an NP-complete computational problem for which the fastest known sequential algorithms require exponential time. The problem became a testbed for the performance of DNA computers after Lipton, [54], demonstrated that it was well suited to take advantage of the parallelism afforded by molecular computation. In 2002, Adleman and his group solved a 20-variable 3-SAT problem, [15]. Unlike the initial proof-of-concept DNA computing experiment, this was the first experiment that demonstrated that DNA computing devices can exceed the computational power of an unaided human. Indeed, the answer to the problem was found after an exhaustive search of more than 1 million (2^{20}) possible solution candidates.

The input to a 3-SAT problem is a Boolean formula in 3CNF, that is, in conjunctive normal form where each conjunct has only at most three literals. This formula is called *satisfiable* if there exists a truth value assignment to its variables that satisfies it, that is, that makes the whole formula true. Thus, the output to the 3-SAT problem is "yes" if such a satisfying truth value assignment exists, and "no" otherwise.

The input formula of Adleman's experiment was the 20-variable, 24-clause, 3CNF formula: $\Phi = (\overline{x_3} \vee \overline{x_{16}} \vee x_{18}) \wedge (x_5 \vee x_{12} \vee \overline{x_9}) \wedge (\overline{x_{13}} \vee \overline{x_2} \vee x_{20}) \wedge (x_{12} \vee \overline{x_9} \vee \overline{x_5}) \wedge (x_{19} \vee \overline{x_4} \vee x_6) \wedge (x_9 \vee x_{12} \vee \overline{x_5}) \wedge (\overline{x_1} \vee x_4 \vee \overline{x_{11}}) \wedge (x_{13} \vee \overline{x_2} \vee \overline{x_{19}}) \wedge (x_5 \vee x_{17} \vee x_9) \wedge (x_{15} \vee x_9 \vee \overline{x_{17}}) \wedge (\overline{x_5} \vee \overline{x_9} \vee \overline{x_{12}}) \wedge (x_6 \vee x_{11} \vee x_4) \wedge (\overline{x_{15}} \vee \overline{x_{17}} \vee x_7) \wedge (\overline{x_6} \vee x_{19} \vee x_{13}) \wedge (\overline{x_{12}} \vee \overline{x_9} \vee x_5) \wedge (x_{12} \vee x_1 \vee x_{14}) \wedge (x_{20} \vee x_3 \vee x_2) \wedge (x_{10} \vee \overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_5} \vee x_9 \vee \overline{x_{12}}) \wedge (x_{18} \vee \overline{x_{20}} \vee x_3) \wedge (\overline{x_{10}} \vee \overline{x_{18}} \vee \overline{x_{16}}) \wedge (x_1 \vee \overline{x_{11}} \vee \overline{x_{14}}) \wedge (x_8 \vee \overline{x_7} \vee \overline{x_{15}}) \wedge (\overline{x_8} \vee x_{16} \vee \overline{x_{10}})$,

where $\overline{x_i}$ denotes the negation of x_i, $1 \leq i \leq 20$. This formula Φ was designed so as to have a unique satisfying truth assignment. The unique solution is: $x_1 = F, x_2 = T, x_3 = F, x_4 = F, x_5 = F, x_6 = F, x_7 = T, x_8 = T, x_9 = F, x_{10} = T, x_{11} = T, x_{12} = T, x_{13} = F, x_{14} = F, x_{15} = T, x_{16} = T, x_{17} = T, x_{18} = F, x_{19} = F, x_{20} = F$.

Adleman's solution was based on the following nondeterministic algorithm.

Input: A Boolean formula Φ in 3CNF.
Step 1: Generate the set of all possible truth value assignments.
Step 2: Remove the set of all truth value assignments that make the first clause false.
Step 3: Repeat Step 2 for all the clauses of the input formula.
Output: The remaining (if any) truth value assignments.

To implement this algorithm, one needs to first encode the input data as DNA strands. This was achieved as follows. Every variable x_k, $k = 1, \ldots, 20$, was associated with two distinct 15-mer DNA single strands called "value sequences." One of them, denoted by X_k^T, represented true (T), while the second, denoted by X_k^F, represented false (F).

The following are some examples of the particular 15-mer sequences—none of which contained the nucleotide G—synthesized and used in the experiment:

$$X_1^T = \text{TTACACCAATCTCTT}, \quad X_1^F = \text{CTCCTACAATTCCTA},$$
$$X_{20}^T = \text{ACACAAATACACATC}, \quad X_{20}^F = \text{CAACCAAACATAAAC}.$$

Each of the possible 2^{20} truth assignments was represented by a 300-mer "library strand" consisting of the ordered catenation of one 15-mer value sequence for each variable, that is, by a strand

$$\alpha_1 \alpha_2 \ldots \alpha_{20}, \text{ where } \alpha_i \in \{X_i^T, X_i^F\}, 1 \leq i \leq 20.$$

To obtain these "library strands," the 40 individual 15-mer sequences (each present in multiple copies in solution) were assembled using the mix-and-match combinatorial synthesis technique of [23].

The biocomputation wetware essentially consisted of a glass "library module" filled with a gel containing the library, as well as one glass "clause module" for each of the 24 clauses of the formula. Each clause module was filled with gel containing probes (immobilized DNA single strands) designed to bind only library strands encoding truth assignments statisfying that clause.

The strands were moved between the modules with the aid of gel electrophoresis, that is, by applying an electric current that resulted in the migration of the negatively charged DNA strands through the gel.

The protocol started with the library passing through the first "clause module," wherein library strands containing the truth assignments satisfying the 1st clause were captured by the immobilized probes, while library strands that did not satisfy the first clause continued into a buffer reservoir. The captured strands were then released by raising the temperature, and used as input to the second clause module, and so on. At the end, only the strand representing the truth assignment that satisfied all the 24 clauses remained.

The output strand was PCR amplified with primer pairs corresponding to all four possible true–false combinations of assignments for the first and last variable x_1 and x_{20}. None except the primer

pair $(X_1^F, WK(X_{20}^F))$ showed any bands, thus indicating two truth values of the satisfying assignment, namely $x_1 = F$ and $x_{20} = F$. The process was repeated for each of the variable pairs (x_1, x_k), $2 \leq k \leq 19$, and, based on the lengths of the bands observed, value assignments were given to the variables. These experimentally derived values corresponded to the unique satisfying assignment for the formula Φ, thus concluding the experiment.

One of the remarkable features of this DNA computing experiment was that simple bio-operations could carry on a complex computation such as the one needed for solving a relatively large instance of 3-SAT.

31.4 Benenson Automata

Following Adleman's, [1], proof-of-concept demonstration of DNA computing, several research teams embarked on the quest for a DNA implementation of a Universal Turing Machine. In the process, they demonstrated various molecular scale autonomous programmable computers ([10–12,56,74]) allowing both input and output information to be in molecular form. In this section we illustrate one such programmable computer, [10], known as the "Benenson automaton."

In [10] the authors construct a simple two-state automaton, Figure 31.1, over a two-letter alphabet set, by using double-stranded DNA molecules and restriction enzymes.

Any particular two-state automaton will have a subset of the eight possible transition rules, and a subset of the two-state set as final states. The automaton implemented by Benenson et al. takes strings of symbols over the two-letter alphabet $\{a, b\}$ as input, and accepts only those strings that contain an even number of letters b (see Figure 31.1).

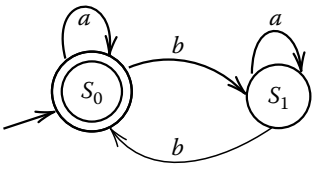


FIGURE 31.1 A finite automaton with two states S_0 and S_1 over the alphabet set $\{a, b\}$, that accepts only words with an even number of letters b.

The main engine of the Benenson automaton is the *FokI* enzyme, a member of an unusual class of restriction enzymes that recognize a specific DNA sequence and cleave nonspecifically a short distance away from that sequence. *FokI* binds the sequence

$$\begin{array}{l} 5' - \text{GGATG} - 3' \\ 3' - \text{CCTAC} - 5' \end{array},$$

and cleaves DNA (regardless of the sequence composition) 9 bp away on the "top" strand, and 13 bp away on the "bottom" strand, leaving thus four-letter-long sticky ends.

The input symbols, a, b, and the terminator t, are encoded respectively as the 6bp sequences:

$$\mathbf{a}: \begin{array}{l} 5' - \text{CTGGCT} - 3' \\ 3' - \text{GACCGA} - 5' \end{array} \quad \mathbf{b}: \begin{array}{l} 5' - \text{CGCAGC} - 3' \\ 3' - \text{GCGTCG} - 5' \end{array} \quad \mathbf{t}: \begin{array}{l} 5' - \text{TGTCGC} - 3' \\ 3' - \text{ACAGCG} - 5' \end{array}$$

An example of the encoding of an input string can be seen in the first line of Figure 31.3. The input ab is encoded as a DNA double strand that contains a restriction site for *FokI*, followed closely by the catenation of the encodings for the symbol string abt.

Every state/symbol pair is encoded as a 4-mer DNA strand in the following way. The state/symbol pair S_0a is encoded as $5' - \text{GGCT} - 3'$ (the 4-mer suffix of the encoding for the symbol a), while S_1a is encoded as $5' - \text{CTGG} - 3'$ (the 4-mer prefix of the encoding for a). The encodings are chosen in a similar way for the state/symbol pairs involving the input symbol b, and the terminator symbol t. This method permits the encoding of a symbol to be interpreted in two ways: If the 4-mer suffix of the encoded symbol is detected, then the symbol is interpreted as being read in state S_0; and if the 4-mer prefix of the encoded symbol is detected, then the symbol is being interpreted as being read in state S_1.

There are two output detection molecules. S_0-D is a 161-mer DNA double strand with an overhang $3' - \text{AGCG} - 5'$, which "detects" the last state of the computation as being S_0. S_1-D is a 251-mer DNA

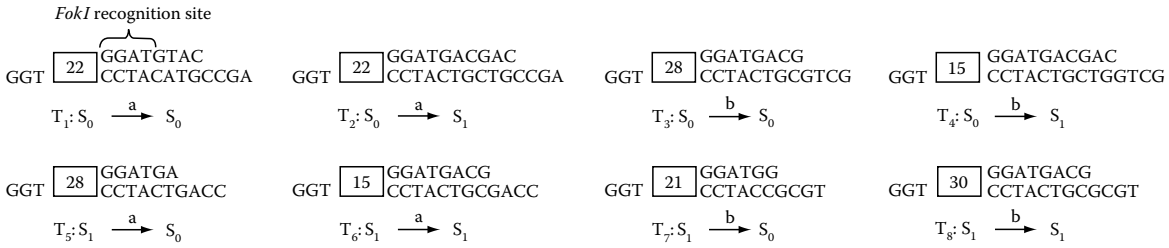


FIGURE 31.2 Transition molecules encoding the set of all possible transition rules of a two-state automaton. The boxes represent DNA double-strands of length equal to the number inside the box. (Adapted from Benenson, Y. et al., *Nature*, 414, 430, 2001.)

double strand with an overhang $3' - ACAG - 5'$, which detects the last state of the computation as being S_1. The two output detection molecules have different lengths, so that they can be easily differentiated by gel electrophoresis.

The eight possible transition molecules of a two-state Benenson automaton are shown in Figure 31.2. Each has a four-letter overhang, for example, $3' - CCGA - 5'$ for T_1, that can selectively bind to DNA molecules encoding the current state/symbol pair, as detailed in the following example.

The automaton processes the encoding for the input string *ab* as shown in Figure 31.3. First, the *FokI* enzyme recognizes its restriction site and cleaves the input encoding the symbols *abt*, thereby exposing the 4-nucleotide sticky-end $5' - GGCT - 3'$ that encodes the state/symbol pair S_0a.

The transition molecule T_1 encoding the automaton rule $S_0a \longrightarrow S_0$ "detects" this state/symbol, by binding exactly to the cleaved input molecule and forming a fully double-stranded DNA molecule with the aid of the enzyme ligase.

The transition molecule T_1, now incorporated in the current molecule, contains a *FokI* recognition site. Moreover, the 3bp "spacer" sequence that follows the *FokI* restriction site in T_1 ensures that the next cleaving of *FokI* will expose a suffix of the encoding of the next input symbol *b*, which will be correctly interpreted as S_0b.*

The overhang of the current DNA molecule is now $5' - CAGC - 3'$, which is interpreted as S_0b. The sticky-end of this sequence fits the transition rule T_4, encoding the automaton rule $S_0b \longrightarrow S_1$. Thus, the combination of the current DNA strand with T_4 and the enzyme ligase leads to another fully double-stranded DNA strand.

A last use of *FokI* exposes the overhang $5' - TGTC - 3'$ of the current DNA molecule. This is a prefix of the terminator, which is interpreted as S_1t. The overhang is complementary to the sticky-end $3' - ACAG - 5'$ of the detector molecule S_1-*D*, corresponding to the last state of the computation being S_1. The state S_1 is not final, and thus the outcome of the computation is that the input *ab* is not accepted by this automaton.

Note that any two-state two-symbol automaton could be built using the aforementioned method. The automaton could be made to perform a different task by choosing a different set of transition molecules.

As an exciting application, Benenson et al. [12] demonstrated how the aforementioned method can be used for medical diagnosis and treatment. As a proof of principle, Benenson et al. [12] programmed an automaton to identify and analyze the mRNA of disease-related genes associated with models of small-cell lung cancer and prostate cancer, and to produce a single-stranded DNA molecule modeled after an anticancer drug.

* Note that, if the automaton would have had instead the transition rule T_2, encoding $S_0a \longrightarrow S_1$, then the length of the "spacer" in T_2 would have been 5bp, placing the position of the cut 2bp more to the left. This would have exposed a prefix of the encoding for *b*, which would have been correctly interpreted as S_1b.

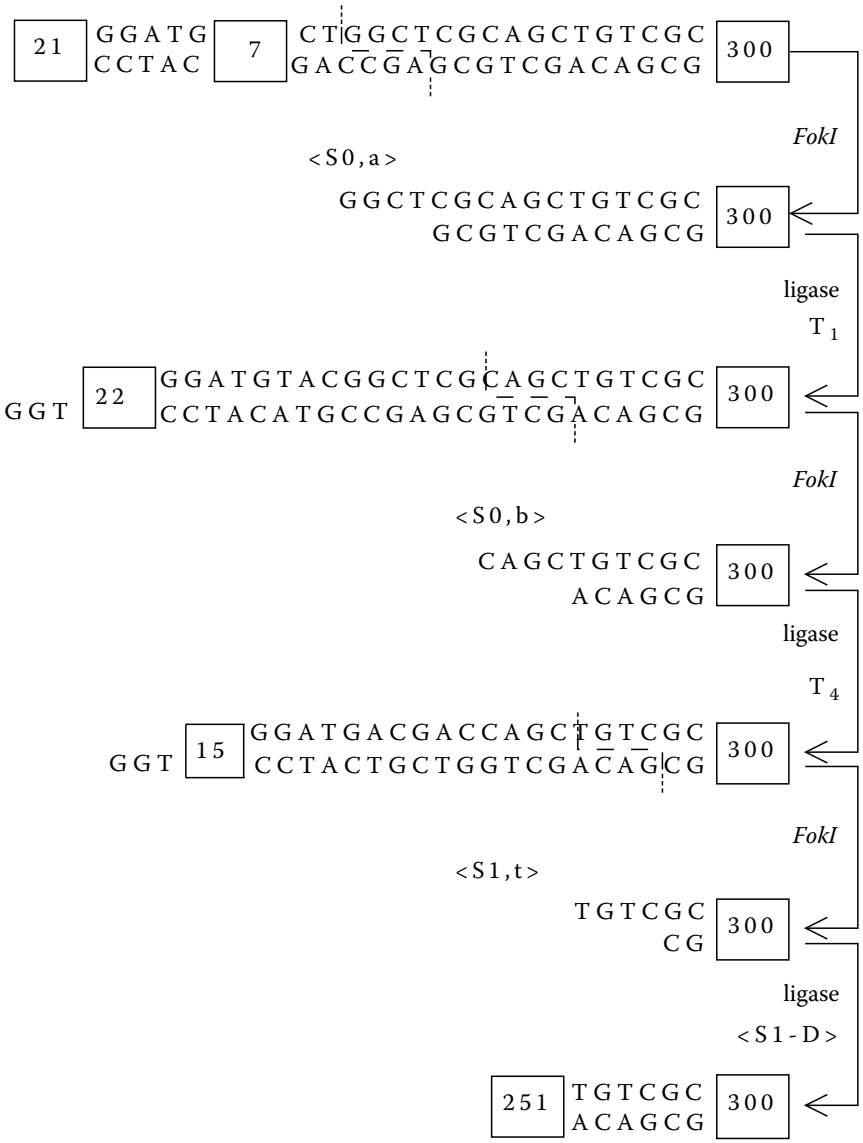


FIGURE 31.3 Example of the computation of a Benenson automaton corresponding to Figure 31.1, for the input *ab*. The numbers in the boxes indicate the lengths of the corresponding double-stranded DNA sequences. (Adapted from Benenson, Y. et al., *Nature*, 414, 430, 2001.)

The computational process of this automaton was formalized in [86]. The authors of [86] study the computational power of Benenson automata and show that they are capable of computing arbitrary Boolean functions.

31.5 DNA Memory

This section discusses various aspects of DNA-encoded information. A brief description of several models for a DNA memory is followed by a more detailed description of the Nested Primer Molecular Memory (NPMM) (Section 31.5.1), and *in vivo* Organic DNA Memory (Section 31.5.2). In addition, Section 31.5.3 provides a discussion on the various approaches to the optimal design of information-encoding DNA sequences for DNA computing, including software simulations, algorithmic methods, and theoretical studies.

There are several reasons to consider DNA memory as an alternative to all the currently available implementations of memories. The first is the extraordinary information-encoding density that can be achieved by using DNA strands. Indeed, the possibility of storing vast amounts of data in a small space is probably one of the most alluring features of DNA computing. In [71], Reif reported a calculation based on the facts that a mole contains 6.02×10^{23} DNA base monomers, and the mean molecular weight of a monomer is approximately 350 g/mole. Thus, 1 g of DNA contains 2.1×10^{21} DNA bases. Since there are 4 DNA bases that can encode 2 bits, it follows that 1 gram of DNA can store approximately 4.2×10^{21} bits. Thus, DNA has the potential of storing data on the order of 10^{10} more compactly than conventional storage technologies. In addition, the robustness of DNA data ensures the maintenance of the archived information over extensive periods of time [4,18,85].

The idea of a content-addressable DNA memory, able to store binary words of fixed length, was first proposed by Baum in [8]. In his model, each word in the memory would be a DNA single strand constructed as follows. Two distinct DNA sequences would be assigned to each component, the first encoding a "1," and the other encoding a "0." DNA molecules encoding a particular binary word would then be obtained by catenating the appropriate DNA sequences corresponding to its bits, in any order. This DNA memory would be associative, or content addressable. Given a subset of the component values, one could retrieve any words consistent with these values from the DNA memory as follows. For each component, one could synthesize the complement of the corresponding subsequence of encoding DNA, and affix it to a solid support, for example, a magnetic bead. This complement would then bond to DNA memory molecules that contain that subsequence, that is, code for that component value. These molecules could then be extracted magnetically. After iterative extractions based on each component, one would end up with DNA molecules matching the constraints exactly. As calculated in [8], the storage that is in principle possible using these techniques is astonishing, making it possible to imagine DNA memories vastly larger than the brain.

Several other authors, [16,46,60,71,88], have proposed various DNA memory models. Almost all of this research consists of preliminary experiments on a very small scale. Recently, Yamamoto et al. proposed a DNA memory with over 10 million addresses [99]. Their proposed DNA memory is addressable by using nested PCR, was named NPMM, and will be discussed in more detail in the Section 31.5.1.

31.5.1 Nested Primer Molecular Memory

NPMM, [99], is a pool of DNA strands wherein each strand codes both data information and its corresponding address information. The data information is a DNA sequence that uses a suitable encoding to express the information as a sequence over the DNA alphabet {A, C, G, T}. The address information consists of several components expressed as DNA sequences flanking the data. For example, data could be stored as

$$[CL_i, BL_j, AL_k, \text{DATA}, AR_q, BR_r, CR_s],$$

where $i, j, k, q, r, s \in \{0, \ldots, 15\}$, and each of the components, for example, CL_0, represents a 20-mer DNA sequence. To retrieve the data, one uses nested PCR consisting of three steps. The first step is to use PCR with the primer pair $(CL_i, WK(CR_s))$. As a result, one can extract from the memory solution the set of DNA molecules starting with CL_i and ending in CR_s. This is because PCR results in a significant difference in concentration between the amplified molecules (starting with CL_i and ending with CR_s), and nonamplified molecules. The latter can be disregarded for all practical purposes. The second PCR is performed with primer pair $(BL_j, WK(BR_r))$. At this point we possess DNA-encoded data that was flanked by CL_iBL_j and BR_rCR_s. The third step is PCR using primer pair

$(AL_k, WK(AR_q))$. This will result in the target DNA-encoding data with "left address" $CL_iBL_jAL_k$ and "right address" $CR_qBR_rAR_s$. The molecules can then be sequenced and decoded to allow the retrieval of the target DNA-encoded data.

The aforementioned NPMM model can realize both enormous address space and high specificity. The hierarchical address structure enables a few DNA sequences to express very large address spaces. One of the main disadvantages of NPMM is that during PCR mutations can occur. This can be avoided by proper selection of DNA sequences, which will be discussed in detail in Section 31.5.3. The authors discuss, in [99], the limitation of scaling up the proposed DNA memory by using a theoretical model based on combinatorial optimization with some experimental restrictions. The results reveal that the size of the address space of this model of DNA memory is close to the theoretical limit.

31.5.2 Organic DNA Memory

The development of a DNA memory technology utilizing living organisms has a much greater potential than any of the existing counterparts to render long life expectancy of data, [97,98]. Since a naked DNA molecule is easily destroyed in any open environment, a living organism can act as a host that protects the information-encoding DNA sequence. In [97], the authors propose a candidate for a living host for DNA memory sequences, that tolerates the addition of artificial gene sequences and survives extreme environmental conditions.

The experiment had several key stages, [97]. In the first stage, in the process of identifying candidates to carry the embedded DNA molecules, the authors considered several microorganisms, and chose two well-understood bacteria, *Escherichia coli*, and *Deinococcus radiodurans*. In particular, *Deinococcus* can survive extreme conditions including cold, dehydration, vacuum, acid, and radiation, and is therefore, known as a polyextremophile.

During the *information encoding* stage, a certain encoding scheme was chosen, that assigned 3-mer sequences to various symbols. For example, AAA encoded the digit "0," AAC the digit "1," and AGG the letter "A" of the English alphabet. Each of the encoding DNA sequences contained only three of the four DNA nucleotides. Using this encoding, any English text could be codified as a DNA sequence, and the text chosen for this experiment was "And the oceans are wide."

In addition, several DNA sequences were chosen to act as "sentinels" and tag the beginning and the end of the encoded messages, for later identification and retrieval. These sequences were chosen by searching the genomes of both *E. coli* and *Deinococcus*, and identifying a set of fixed-size DNA sequences that do not exist in either genome, yet satisfy all the genomic constraints and restrictions. Twenty five such DNA sequences, 20 base-pairs each, were selected, in such a way so as not to cause unnecessary mutation or damage to the bacteria. All sequences contained multiple stop codons TAA, TGA, and TAG as subsequences. Without their presence, the bacterium could misinterpret the memory strands, transcribe them into mRNA, and then translate the mRNA into artificial proteins that could destroy the integrity of the embedded message, or even kill the bacterium.

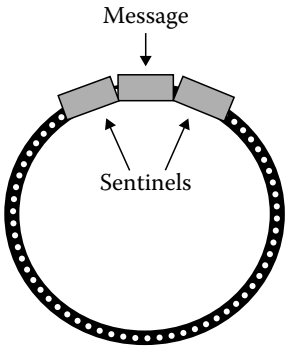


FIGURE 31.4 A recombinant plasmid with two DNA fragments as sentinels protecting the DNA-encoded message in between. (Reproduced from Wong, P. et al., *Commun. ACM*, 46, 95, 2003.)

A 46bp DNA sequence was then created, consisting of two different 20bp selected "sentinel" sequences, connected by a 6bp sequence that was the recognition site of a particular enzyme. This double-stranded DNA molecule was cloned into a recombinant plasmid (a circular double-stranded DNA sequence)—see Figure 31.4.

The embedded DNA was then inserted into cloning vectors (circular DNA molecules that can self-replicate within a bacterial host). The resultant vectors were then transferred into *E.coli* by high-voltage shocks, allowing the vector to multiply.

The vector and the encoded DNA were then incorporated into the genome of *Deinococcus* for permanent information storage and retrieval. Using prior knowledge of the sequences at both borders, the message was retrieved by PCR, read-out, and decoded into the original English text it encoded.

The proposed organic DNA memory has an enormous potential capacity for storing information, especially considering that 1mL of liquid can contain up to 10^9 bacteria. Potential disadvantages are random mutations, but these are unlikely given the well developed natural mechanisms that exist in cells for detecting and correcting errors.

31.5.3 Design of DNA Sequences

In most DNA-based computations, there are three basic stages: (1) encoding the input data using single- or double-stranded DNA molecules, (2) performing the biocomputation using bio-operations, and finally (3) decoding the result. One of the main problems in DNA computing is associated with step (1), and concerns the design of optimal information-encoding oligonucleotides to ensure that mismatched pairing due to the Watson–Crick complementarity is minimized, and that undesirable bonds between DNA strands are avoided.

Indeed, in laboratory experiments, the complementarity of the bases may pose potential problems if some DNA strands form nonspecific hybridizations and partially anneal to strands that are not their exact complements, or if they stick to themselves or to each other in unintended ways. There are several approaches to addressing this so-called *sequence design problem*. In this chapter, we discuss the software simulation approach, the algorithmic approach, and the theoretical approach to the design of DNA strands optimal for DNA computing.

31.5.3.1 Software Simulation

Software simulation tools verify the computation protocol correctness before it is carried out in a laboratory experiment. Several software packages [24,25,32,33] written for DNA computing purposes are available. For example, the software *Edna* simulates biochemical processes and reactions that can occur during a laboratory experiment.

Edna, [28], is a simulation tool that uses a cluster of PCs and simulates the processes that could happen in test tubes. *Edna* can be used to determine if a particular choice of encoding strategy is appropriate, to test a proposed protocol and estimate its performance and reliability, and even to help assess the complexity of the protocols. Test tube operations are assigned a cost that takes into account many of the reaction conditions. The measure of complexity used by *Edna* is the sum of the costs added up over all operations in a protocol. Other features offered by the software allow the prediction of DNA melting temperature (the temperature at which a DNA double strand "melts" into its two constituent single strands), taking into account various reaction conditions. All molecular interactions simulated by the software are local and reflect the randomness inherent in biomolecular processes.

31.5.3.2 Algorithmic Method

In most DNA-based computations there is an assumption that a strand will bind only to its perfect Watson–Crick complement. For example, the results of DNA computations are retrieved from the test tubes by using strands that are complementary to the ones used for computation.

In practice, it is, however, possible that a DNA molecule will bind to another molecule, which differs from its perfect complementary molecule by one or even several nucleotides. One way to

avoid this is to ensure that two molecules in the solution differ in more than one location. This property can be formalized in terms of the Hamming distance.

Given two words w_1, w_2 of equal length, the Hamming distance $H(w_1, w_2)$ is defined as the number of locations in which the words w_1 and w_2 are distinct. For a set of words, the Hamming distance constraint requires that any two words w_1 and w_2 in the set have $H(w_1, w_2) \geq d$. If we are dealing with DNA words, another constraint necessary to avoid mishybridizations is $H(w_1, WK(w_2)) \geq d$, where WK(w) denotes the Watson–Crick complement of w. Yet another consideration is that, when retrieving the results from the solution, hybridization should occur simultaneously for all molecules in the solution. This implies that the respective melting temperatures should be comparable for all hybridization reactions that are taking place. This is the third main constraint that the set of words under consideration needs to adhere to.

To address the problem of designing DNA code words according to these three constraints, an algorithm based on a stochastic local search method was proposed in [89]. The melting temperature constraint is simplified to the constraint requiring that, for each strand, the number of C and G nucleotides amounts to 50% of its total nucleotide count. The algorithm produces a set of DNA sequences of equal length that satisfy the Hamming distance and the temperature constraints. The algorithm is based on the following:

Input: Number k of words needed to produce, and the word length n.
Step 1: Produce a random set of k words of length n.
Step 2: Modify the set so that the set satisfies the first constraint.
Step 3: Repeat Step 2 for all the given constraints.
Output: The set of words (if one can be found).

More specifically, for *Step 2* two words w_1 and w_2 are chosen from the set that violate at least one of the constraints. With a probability $1 - \theta$, θ being the noise parameter, one of these words is altered by randomly substituting one base in a way that maximally decreases the number of constraint violations. The algorithm terminates either when there are no more constraint violations in the set of words or when the number of loop iterations has exceeded some maximum threshold. Empirical results prove this technique to be effective and the noise parameter θ is empirically determined to be optimal as 0.2, regardless of the problem instance.

31.5.3.3 Theoretical Studies

In this section, we discuss the formal language theoretical approach to the problem of designing code words, introduced in [35]. We begin by reviewing some basic notions and notations. An alphabet is a finite, nonempty set of symbols. Let Σ be such an arbitrary alphabet. Then Σ^* denotes the set of all words over this alphabet, including the empty word λ. Σ^+ is the set of all nonempty words over Σ. Σ^i is the set of all words over Σ of length i. We also denote by $\text{Sub}_k(L)$, the set of all subwords of length k of words from the set L. For more details of formal language theory, theory of codes, and combinatorics on words the reader is referred to [14,34,55,75,83,102].

Every biomolecular protocol involving DNA generates molecules whose sequences of nucleotides form a language over the four-letter alphabet $\Delta = \{A, G, C, T\}$. The Watson–Crick complementarity of the nucleotides defines a natural involution θ, $A \mapsto T$ and $G \mapsto C$ which is an antimorphism on Δ^*. An involution θ is a mapping such that θ^2 is the identity. An antimorphism θ is such that $\theta(uv) = \theta(v)\theta(u)$ for all words u, v from Δ^*. For example, if $u =$ AGACT, then the Watson–Crick complement $\theta(u)$ is AGTCT.

Given a DNA language, that is, a language over the DNA alphabet Δ, undesirable Watson–Crick bonds between its words can be avoided if the language satisfies certain properties. There are two types of unwanted hybridizations: intramolecular and intermolecular. The intramolecular

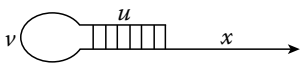


FIGURE 31.5 Intramolecular hybridization (hybridization within the same molecule). A DNA sequence of the type $uv\overleftarrow{u}x$, where $\overleftarrow{u}$ denotes the Watson–Crick complement of u, will form a secondary structure called a hairpin. Such hairpin structures are avoided by hairpin-free languages and by θ-k-m-subword codes.

hybridization happens when two sequences, one being the reverse complement of the other, appear within the same DNA strand (see Figure 31.5). In this case the DNA strand forms a hairpin.

Suppose we want to avoid the type of hybridization shown in Figure 31.5. A language L is called (Jonoska et al., [36]) a θ-k-m-subword code if for all words $u \in \Sigma^k$ we have $\Sigma^* u \Sigma^m \theta(u) \Sigma^* \cap L = \emptyset$. An analogous definition of hairpin-free languages was given in [41]. In this definition, Σ is an arbitrary alphabet and θ is any antimorphic involution. If Σ is taken to be the DNA alphabet Δ, then this property is essentially saying that the DNA sequences from the set L do not form the hairpin structures illustrated in Figure 31.5.

DNA strand sets that avoid all types of unwanted intermolecular bindings (See Figure 31.6) were introduced in [36], and called θ-k-codes. A language L is said to be a θ-k-code if $\theta(x) \neq y$ for all $x, y \in \text{Sub}_k(L)$. The relationship $\theta(x) = y$ indicates that the molecules corresponding to x and y can form chemical bonds between them as shown in Figure 31.6. For a suitable k, a θ-k-code avoids all kinds of unwanted intermolecular hybridizations.

The θ-k-code property is meant to ensure that DNA strands do not form unwanted hybridizations during DNA computations. Sets theoretically designed to have this property have been successfully tested in practical wet-lab experiments [36]. In [44], the concept of θ-k-code has been extended to the Hamming bond-free property: $H(\theta(x), y) > d$ for any subwords $x, y \in \text{Sub}_k(L)$, where d is an empirically chosen positive integer parameter.

Suppose we use codes that have the language properties we have described, what may happen during the course of computation is that the properties initially present deteriorate over time. This leads to another study, which investigates how bio-operations such as cutting, pasting, splicing, contextual insertion, and deletion affect the various bond-free properties of DNA languages. Invariance under these bio-operations has been studied in [36,37,40]. Bounds on the sizes of some other codes with desirable properties that can be constructed are explored by Marathe et al. [57].

The approach to DNA-encoded information and its properties described in this section has been meaningful to both DNA computing experimental research, and to DNA computing theoretical research by establishing a mathematical framework for describing and reasoning about DNA-encoded information. In addition, the notions defined and investigated in this context proved to be fruitful from the purely computer science point of view. Indeed, it turns out that some of these notions generalize classical concepts in coding theory and combinatorics of words such as primitive,

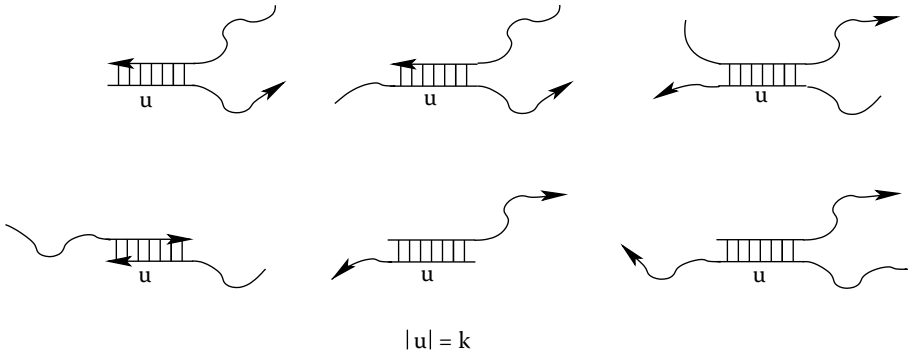


FIGURE 31.6 Intermolecular hybridizations (hybridizations between different molecules): If one DNA molecule contains a subword u of length k, and the other one contains the subword $\leftarrow u$, then the molecules will bind to each other, as shown in this figure. Such hybridizations are avoided by θ-k-subword codes.

commutative, conjugate, and palindromic words, as well as prefix, suffix, infix, and comma-free codes (see, e.g., [19,35,40,42,43]).

31.6 Computation in Living Cells

An example of the attempts to understand nature as computation is the research on computation in and by living cells. This is also sometimes called cellular computing, or *in vivo* computing, and one particular area of study in this field concerns the computational capabilities of gene assembly in unicellular organisms called ciliates [22,50].

Ciliates, unicellular protozoa named for their wisp-like cover of cilia, possess two types of nuclei: an active macronucleus containing the functional genes, and a functionally inert micronucleus that contributes only to genetic information exchange. In the process of conjugation, after two ciliates exchange genetic information and form new micronuclei, they have to assemble in real-time new macronuclei necessary for their survival. This is accomplished by a process that involves reordering some fragments of DNA from the micronuclear DNA (permutations and some inversions) and deleting other fragments. The process of obtaining the macronuclear DNA from the micronuclear DNA, by removing the noncoding sequences and permuting the coding fragments to obtain the correct order, is called *gene rearrangement* [66] (see Figure 31.7). The function of the various noncoding eliminated sequences is unknown and they represent a large portion (up to 98%) of the micronuclear sequences. As an example, the micronuclear *Actin I* gene in *Oxytricha nova* is composed of 9 coding segments present in the permuted order 3-4-6-5-7-9-2-1-8, and separated by 8 noncoding sequences. Instructions for the gene rearrangement (also called gene unscrambling) are apparently carried in the micronuclear DNA itself: pointer sequences present at the junction between coding and noncoding sequences, as well as certain RNA "templates," permit reassembly of the functional macronuclear gene.

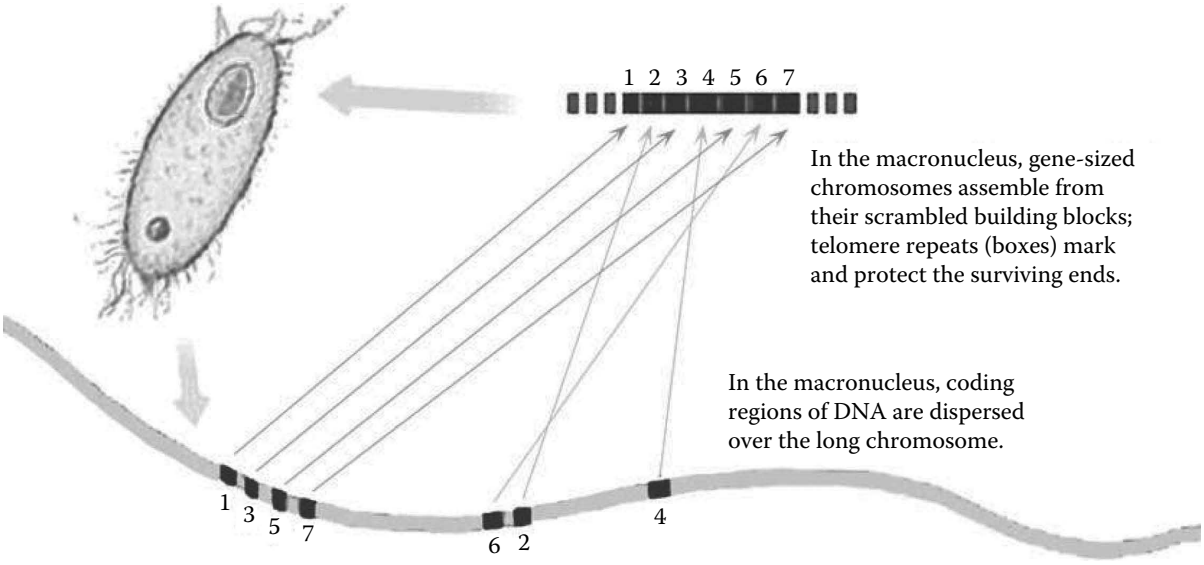


FIGURE 31.7 Overview of gene rearrangement in ciliates. Dispersed micronuclear coding fragments 1–7 (bottom) reassemble during macronuclear development to form the functional gene copy (top). (From Landweber, L. and Kari, L., The evolution of cellular computing: Nature's solution to a computational problem, *Proceedings of the DNA 4*, University of Pennsylvania, Philadelphia, PA (Reproduced from L. Kari, H. Rubin, and D. Wood, Eds.); *Biosystems*, 52, 3, 1999.)

The process of gene assembly is interesting from both the biological and the computational point of view. From the computational point of view, this study led to many novel and challenging research themes, [22]. Among others, it was proved that various models of gene assembly have full Turing machine capabilities, [50]. From the biological point of view, the joint effort of computer scientists and biologists led to a plausible hypothesis (supported already by some experimental data) about the "bioware" that implements the process of gene assembly, which is based on the new concept of template-guided recombination, [3,63,67].

Other approaches to cellular computing include developing of an *in vivo* programmable and autonomous finite-state automaton within *E. coli*, [59], and designing and constructing *in vivo* cellular logic gates and genetic circuits that harness the cell's existing biochemical processes [49,92,93].

31.7 DNA Self-Assembly

One of the most important achievements of DNA computing has been its contribution to the extensive body of research in nanosciences, by providing computational insights into a number of fundamental issues. Perhaps, the most prominent is its contribution to the understanding of self-assembly, which is among the key concepts in nanosciences (see [70] for an overview of self-assembly written for computer scientists). Self-assembly originates in the formal two-dimensional model of tiling defined and studied by Wang, [90]. Tiles are squares with "coloured edges" that, when placed on the plane, form a valid tiling only if the colours at their abutting edges are equal. Tiling systems can be used to simulate the computation of a universal Turing Machine (see, e.g., [13,90]). Winfree et al. [94,95], were the first to implement the concepts of computational tiling by using self-assembling "DNA tiles." This section describes several models of self-assembly, as well as applications of self-assembly to computation, (Section 31.7.1), to the creation of static DNA shapes and patterns (Section 31.7.2), as well as to engineering dynamic DNA nanomachines (Section 31.7.3).

31.7.1 DNA Self-Assembly for Computation

Molecular self-assembly is an automatic process in which molecules assemble into covalently bonded, well-defined stable structures without guidance or management from an outside source. There are two types of self-assembly, namely, intramolecular self-assembly and intermolecular self-assembly. Most often the term molecular self-assembly refers to intermolecular self-assembly, while the intramolecular analog is more commonly called folding.

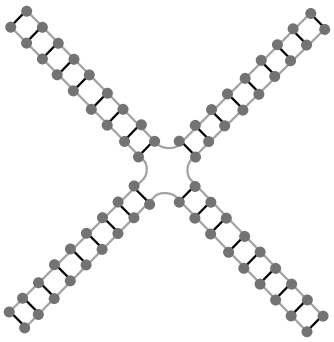


FIGURE 31.8 DNA Holiday junction. Four DNA single strands can be designed so as to form a branched structure in which each single strand participates in two consecutive double-stranded "branches."

One of the important features of DNA used in self-assembly is that DNA has many rigid, well-characterized forms that are not a linear double helix. For example, one can build DNA structures called Holiday junctions, [77], wherein four DNA single strands self-assemble and create a branched structure, as seen in Figure 31.8. This can be accomplished by designing each of the four DNA single strands to be partially complementary to one of the others, so that each strand participates in two consecutive double-stranded "branches."

This type of complex self-assembled DNA structures have been used by Winfree et al. who introduced a two-dimensional self-assembly model for DNA computation, [95]. In this model, using techniques similar to the ones described earlier (demonstrated, e.g., in [27]), rectangular DNA structures are built, with four sticky single-stranded

ends at their corners. These DNA structures behave as "tiles" and can be designed to perform two-dimensional assembly as follows. Depending on their composition, the sticky ends of a tile (which are single-stranded DNA strands) will attach only to Watson–Crick complementary corresponding ends from other tiles. This results in spontaneous self-assemblies of the tiles into essentially planar conformations that can be designed so as to perform any deterministic or nondeterministic computation.

Yokomori et al., [101], introduced another self-assembly model that is Turing-universal. Other computations that have been achieved by the self-assembly of complex DNA nanostructures include bit-wise cumulative XOR, [56], and binary counters [5]. As another example, Seelig et al. [76] reported the design and experimental implementation of DNA-based digital logic circuits. The authors demonstrate AND, OR, and NOT gates, signal restoration, amplification, feedback, and cascading.

Regarding intramolecular self-assembly, Hagiya et al. [31] and Sakamoto et al. [74], proposed a new method of DNA computing that involves a self-acting DNA molecule containing, on the same strand, the input, program, and working memory. The computation, also called Whiplash PCR, proceeds as follows (see Figure 31.9). The single DNA strand contains at its 5′ end state transitions of the type $A \longrightarrow B$, each encoded as a DNA rule block "$\overleftarrow{B} - \overleftarrow{A}$ $-$*stopper sequence*." The 3′ end of the DNA sequence contains, at any given time, the encoding of the "current state," say A. In Step (1), cooling the solution will lead the 3′ end of the DNA strand, A, to attach to its complement in the corresponding rule block, namely to $\overleftarrow{A}$. In Step (2), PCR is used to extend the now-attached end A by the encoding of the new state B, and the extension process is stopped by the stopper sequence. Then, in Step (3), by raising the temperature, the new current state B detaches and becomes loose, and the next state transition cycle can begin.

The main motivation for using this model is that self-acting molecules can compute in parallel, in a single-tube reaction, allowing for a multiple program, multiple input architecture. Hagiya et al., [31], showed how to theoretically learn μ-formulas using this model. Subsequently, Winfree [96] showed how to solve several NP-complete problems in $O(1)$ biosteps using whiplash PCR.

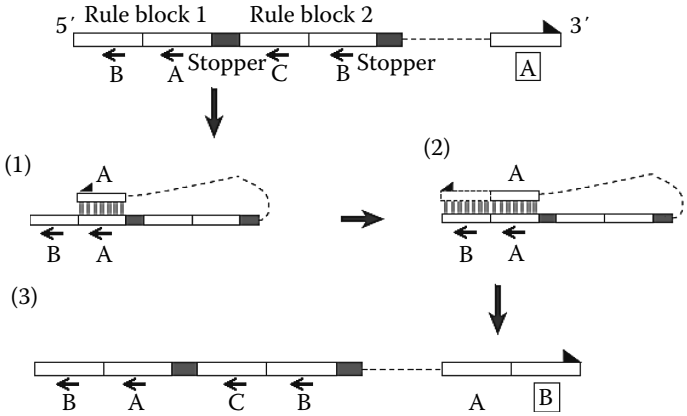


FIGURE 31.9 Whiplash PCR ($\overleftarrow{A}$ denotes the Watson–Crick complement of A). The 5′ end of the DNA strand contains rule blocks "$\overleftarrow{B} - \overleftarrow{A}$ $-$ stopper sequence" coding for state transitions of the type $A \rightarrow B$. The 3′ end contains the encoding of the current state A. In Step (1), cooling the solution leads the 3′ end A to attach to the corresponding part of the rule block. In Step (2), PCR extends the now-attached end encoding the state A, by the new state B, stopping at the stopper sequence. Raising the temperature results, in Step (3), in the loose end encoding the new state B becoming detached and ready for the next state transition cycle.

31.7.2 DNA Nanoscale Shapes

DNA nanotechnology (see [78] for a comprehensive survey), uses the unique molecular recognition properties of DNA to create intricate structures such as self-assembling branched DNA complexes with useful properties.

DNA is a building block ideally suited for nanostructures, as it combines self-assembly properties with programmability. In nanotechnology, DNA is used as a structural material rather than as a carrier of biological information. Using DNA, impressive nanostructures have been created, including two- and three-dimensional structures. Examples include DNA nanostructures such as Sierpinski triangles [73], and cubes [17,78].

Another design theme was introduced in [81]. The authors report that a single strand of DNA (1669 nucleotides long) was folded into a highly rigid nanoscale octahedron structure, of a diameter of about 22 nanometers. The long strand of DNA was designed so as to have a number of self-complementary regions, which would induce the strand to fold back on itself to form a sturdy octahedron. Folding the DNA into the octahedral structure simply required the heating and the cooling of the solution containing the DNA, magnesium ions, and five 40-mer accessory DNA sequences. Moreover, this design permitted the assembly of the first clonable DNA three-dimensional nanostructure.

Rothemund, [72], presented a simple model that uses several short strands of DNA to direct the folding of a long single strand of DNA into desired shapes. The author demonstrates the generality of the method, called "scaffolded DNA origami," that permits the fabrication (out of DNA) of any two-dimensional shape roughly 100 nm in diameter. The techniques used are similar to the ones used in the design of the Holiday junction (Section 31.7), in that the constituent DNA strands form complex structures by their design, which makes it possible for some single strands to participate in two DNA helices—they wind along one helix, then switch to another.

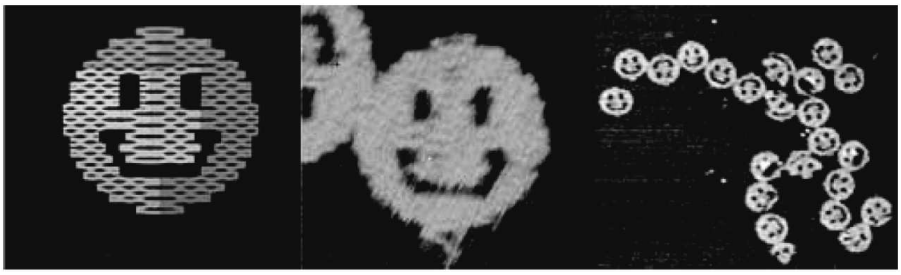
The design process of a DNA origami involves several steps. The first step is to build an approximate geometric model of the desired shape. The shape is approximated by cylinders that are models of the DNA double-helices that will ultimately be used for the construction. The second step is to fill the shape by folding a single long "scaffold strand" back and forth in a raster pattern such that, at each moment, the scaffold strand represents either the "main" strand or the "complement" strand of a double helix. (The process is analogous to drawing out a shape using a single line, and without taking the pencil off the paper.) Once the geometric model and the folding path have been designed, the third step is to use a computer program to generate a set of "staple strands" that provide Watson–Crick complements for the scaffold. The staple strands are designed to bind to portions of the scaffold strand, holding it thus together into the desired shape. The staple strands are then fine-tuned to minimize strain in the construction and optimize the binding specificity and binding energy.

To test this method, Rothemund [72] used as a scaffold circular genomic DNA (7249 nucleotide long) from the virus *M13mp18*. Approximately 250 short staple strands were synthesized and mixed with the scaffold, in 100-fold excess to it. The strands annealed in less than 2 h and AFM (Atomic Force Microscopy) imaging showed that indeed the desired shape was realized. The generality of the method was proved by assembling six different shapes such as squares, triangles, five-pointed stars, and smiley faces (Figure 31.10).

This method not only provides access to structures that approximate the outline of any desired shape, but also enables the creation of structures with arbitrary-shaped holes or surface patterns. In addition, it seems likely that scaffolded DNA origami can be adapted to design three-dimensional structures.

31.7.3 DNA Nanomachines

In addition to static DNA nanoscale shapes and patterns, an impressive array of ingenious DNA nanomachines (see [6,53] for comprehensive reviews), were designed with potential uses to



(a) (b) (c)

FIGURE 31.10 (a) A folding path that creates a "DNA smiley." An even number of double helices are filled into the desired shape. (b), (c) A long single-stranded DNA self-assembles into the desired shape (Atomic Force Microscopy images). (Reproduced from Rothemund, P., *Nature*, 440, 297, 2006.)

nanofabrication, engineering, and computation. DNA nanomachines, that is, DNA-based nanodevices that convert static DNA structures into machines that can move or change conformation, have been developed rapidly since one of the earliest examples was reported in 1998, [100]. They include molecular switches that can be driven between two conformations [52], DNA "tweezers" [103], DNA "walkers" that can be moved along a track [80,82] and autonomous molecular motors [7,30,69].

We illustrate some of the construction principles used with the construction of a device referred to as "molecular tweezers," [103]. The DNA tweezers consist of two partially double-stranded DNA arms connected by a short single DNA strand acting as a flexible hinge (Figure 31.11). The resulting structure is similar in form to a pair of open tweezers. A so-called set strand is designed in such a way as to be complementary to both single stranded "tails" at the end of the arms. The addition of this set strand to the mixture will result in its annealing to both tails of the arms, bringing thus the arms of the tweezers together in a "closed" configuration. A short region of the set strand remains single stranded even after it hybridizes to the arms. This region is used as a toehold that allows a new "reset strand" to strip the set strand from the arms, by hybridizing with the set strand itself. The tweezers are thus returned to the "open" configuration. A number of variations on the tweezers have also been reported [21,26,58,84].

Considerable efforts have been expended on realizing "walking devices." Several molecular machines, which can walk along a track have been proposed [65,79,80,82,87]. Shin and Pierce [82],

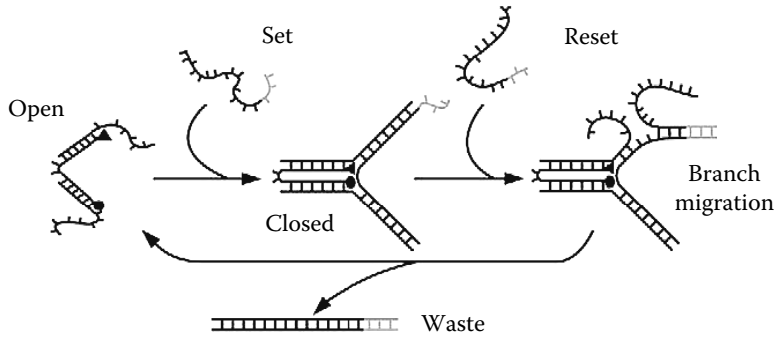


FIGURE 31.11 A DNA nanomachine, "DNA tweezers," driven by repeated sequential additions of "set" and "reset" strands. In the open state, the DNA tweezer consists of two double helical arms (with short single-stranded tails), connected by a single stranded hinge. Hybridization of the tails with the set strand closes the arms. The set strand can be removed via branch migration, by which the fully complementary "reset strand" strips the set strand from the tweezer tails, returning it to the open state. (Reproduced from Liedl, T. et al., *Nanotoday*, 2, 36, 2007.)

introduced a DNA device with two distinguishable feet that "walks" directionally on a linear DNA track with four distinct single strands periodically protruding from it and acting as anchors. The walker is double-stranded and has two single-stranded extensions acting as "legs." Specific attachment strands bind the legs to the single-stranded extensions anchors periodically placed along the double-stranded track. Each step requires the sequential addition of two strands: the first lifts the back foot from the track, by strand displacement—a process by which an invading single DNA strand can "displace" one of the constituent strands of a double-strand, by replacing it with itself, provided the newly formed structure is more stable. The second strand places the released foot ahead of the stationary foot. Brownian motion provides movement, and the order of adding the "instruction" strands ensures directionality. A similar walking device, based on the movement pattern of inchworms, was developed by Sherman and Seeman, [80]: here the front foot steps forward and the back foot catches up.

The walkers described earlier need "fuel DNA," and cannot walk autonomously. Recently, [79] suggested a three-leg molecular walking machine that can walk autonomously in two or three dimensions on a designed route. It uses an enzyme as a source of power, and a track of DNA equipped with many single-stranded DNA anchors arranged in a certain pattern.

We end this section by mentioning research on DNA nanodevices in conjunction with the cell's genetic mechanisms. Cellular organisms exhibit complex biochemical interaction networks comprising genes, RNA and proteins. These include gene activation and inhibition, gene transcription into RNA, and the translation of RNA into proteins. By adapting parts of these genetic mechanisms, one can engineer novel molecular machinery *in vitro* and potentially *in vivo*, [53]. Several attempts have been made to use genetic mechanisms to control DNA-based nanodevices, [47,48,61,62]. For example, pioneering work by Noireaux, Bar-Ziv and Libchaber, [61,62], demonstrates the principles of "cell-free genetic circuit assembly," wherein the term cell-free refers to the fact that the experiments were performed *in vitro*, not *in vivo*. Overall, the successful combination of DNA-based nanodevices with genetic machinery points to promising possible applications to biotechnology, bioengineering, and medicine.

31.8 Conclusion

The excitement that the first DNA computing experiment created in 1994 was primarily due to the fact that computing with DNA offered a completely new way of both looking at and performing computations: by cutting and pasting DNA strands using enzymes, by using the Watson–Crick complementarity of DNA strands, by selecting DNA strands containing a certain pattern, and so on. This novel way of viewing computation has the potential to change the very meaning of the word "compute," and DNA computing has already made significant contributions to the field of computer science. At the same time, research into the computational abilities of cellular organisms has the potential to uncover the laws governing biological information and to enable us to harness the computational power of cells.

This chapter was intended to give the reader a snapshot of DNA computing research by addressing both theoretical and experimental aspects, both "classical" and very recent trends, the impact of molecular computation on the theory and technology of computation, as well as on understanding of the basic mechanisms of bioprocesses.

DNA computing is a rapidly developing field and the recent years already saw some interesting new developments. Experimental advances include self-assembly of cylinders and Möbius strips [29], and new designs for DNA nanomachines, fueled by light, [51]. Progress in the theory of DNA computing include studies of pseudoknot-free DNA/RNA words, [45], of the time-optimal self-assembly of three-dimensional shapes [9], and the proposal of a simple scalable DNA-gate motif for the purpose of synthesizing large-scale circuits [68].

It is envisaged that research into *in vitro* and *in vivo* DNA computing constitutes just a preliminary step that may ultimately lead to molecular computing becoming a viable complementary tool for computation, as well as providing more insight into the computational nature of bioprocesses taking place in living organisms.

Acknowledgments

This research was supported by Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant and Canada Research Chair Award to L. K. We thank Bo Cui, Elena Czeizler, Eugen Czeizler, Zachary Kincaid, Shinnosuke Seki, and the anonymous referee for their suggestions and comments, and Yan Zeng for Figures 31.9 and 31.8.

References

1. L. Adleman, Molecular computation of solutions to combinatorial problems, *Science* 266 (1994) 583–585.
2. M. Amos, *Theoretical and Experimental DNA Computation*, Springer Verlag, Berlin, *Natural Computing Series* 2005.
3. A. Angeleska, N. Jonoska, M. Saito, L. Landweber, RNA-guided DNA assembly, *Journal of Theoretical Biology* 248 (2007) 706–720.
4. C. Bancroft, T. Bowler, B. Bloom, C. Clelland, Long-term storage of information in DNA, *Science* 293 (2001) 1763–1765.
5. R. Barish, P. Rothemund, E. Winfree, Two computational primitives for algorithmic self-assembly: Copying and counting, *Nanoletters* 5 (2005) 2586–2592.
6. J. Bath, A. Turberfield, DNA nanomachines, *Nature Nanotechnology* 2 (2007) 275–284.
7. J. Bath, S. Green, A. Turberfield, A free-running DNA motor powered by a nicking enzyme, *Angewandte Chemie International Edition* 44 (2005) 4358–4361.
8. E. Baum, How to build an associative memory vastly larger than the brain, *Science* 268 (1995) 583–585.
9. F. Becker, E. Remila, N. Schabanel, Time optimal self-assembling of 2D and 3D shapes: The case of squares and cubes, *Prelim. Proc. of DNA 14* (*DNA n* denotes the main international conference in the field of DNA Computing, *The nth International Meeting on DNA Computing.*), Prague, Czech Republic (A. Goel, F. Simmel, P. Sosik, Eds.), June (2008) pp. 78–87.
10. Y. Benenson, T. Paz-Elizur, R. Adar, E. Keinan, Z. Livneh, E. Shapiro, Programmable and autonomous computing machine made of biomolecules, *Nature* 414 (2001) 430–434.
11. Y. Benenson, R. Adar, T. Paz-Elizur, Z. Livneh, E. Shapiro, DNA molecule provides a computing machine with both data and fuel, *Proceedings of the National Academy of Sciences U S A* 100 (2003) 2191–2196.
12. Y. Benenson, B. Gil, U. Ben-Dor, R. Adar, E. Shapiro, An autonomous molecular computer for logical control of gene expression, *Nature* 429 (2004) 423–429.
13. R. Berger, *Undecidability of the domino problem*, *Memoirs of the American Mathematical Society* 66 (1966), 72 pp.
14. J. Berstel, D. Perrin, *Theory of Codes*, Academic Press Inc. Orlando, FL, 1985.
15. R. Braich, N. Chelyapov, C. Johnson, P. Rothemund, L. Adleman, Solution of a 20-variable 3-SAT problem on a DNA computer, *Science* 296 (2002) 499–502.
16. J. Chen, R. Deaton, Y. Wang, A DNA-based memory with in vitro learning and associative recall, *Natural Computing* 4 (2005) 83–101.
17. J. Chen, N. Seeman, Synthesis from DNA of a molecule with the connectivity of a cube, *Nature* 350 (1991) 631–633.

18. J. Cox, Long-term data storage in DNA, *Trends in Biotechnology* 19 (2001) 247–250.
19. E. Czeizler, L. Kari, S. Seki, On a special class of primitive words, *Proceedings of Mathematical Foundations of Computer Science (MFCS)*, Torun, Poland, *LNCS* 5162 (2008) pp. 265–277.
20. M. Daley, L. Kari, DNA computing: Models and implementations, *Comments on Theoretical Biology* 7 (2002) 177–198.
21. B. Ding, N. Seeman, Operation of a DNA robot arm inserted into a 2D DNA crystalline substrate, *Science* 314 (2006) 1583–1585.
22. A. Ehrenfeucht, T. Harju, I. Petre, D. Prescott, G. Rozenberg, *Computation in Living Cells: Gene Assembly in Ciliates*, Springer Verlag, Berlin, *Natural Computing Series*, 2004.
23. D. Faulhammer, A. Cukras, R. Lipton, L. Landweber. Molecular computation: RNA solutions to chess problems, *Proceedings of the National Academy of Sciences U S A* 97 (2000), 1385–1389.
24. U. Feldkamp, W. Banzhaf, H. Rauhe, A DNA sequence compiler, *Preliminary Proceedings of DNA 6*, Leiden, Netherlands (A. Condon, G. Rozenberg, Eds.), June (2000).
25. U. Feldkamp, S. Saghafi, W. Banzhaf, H. Rauhe, DNA sequence generator: A program for the construction of DNA sequences, *Proceedings of DNA 7*, Tampa, FL (N. Jonoska, N. Seeman, Eds.), *LNCS* 2340 (2002) pp. 23–32.
26. L. Feng, S. Park, J. Reif, H. Yan, A two state DNA lattice switched by DNA nanoactuator, *Angewandte Chemie International Edition* 42 (2003) 4342–4346.
27. T. Fu, N. Seeman, DNA double cross-over structures, *Biochemistry* 32 (1993) 3211–3220.
28. M. Garzon, C. Oehmen, Biomolecular computation in virtual test tubes, *Proceedings of DNA 7*, Tampa, FL (N. Jonoska, N. Seeman, Eds.), *LNCS* 2340 (2002) pp. 117–128.
29. M. Gopalkrishnan, N. Gopalkrishnan, L. Adleman, Self-assembly of cylinders and Möbius strips by DNA origami, *Preliminary Proceedings of DNA 14*, Prague, Czech Republic (A. Goel, F. Simmel, P. Sosik, Eds.), June (2008) 181.
30. S. Green, D. Lubrich, A. Turberfield, DNA hairpins: Fuel for autonomous DNA devices, *Biophysical Journal* 91 (2006) 2966–2975.
31. M. Hagiya, M. Arita, D. Kiga, K. Sakamoto, S. Yokoyama, Towards parallel evaluation and learning of boolean μ-formulas with molecules, *Proceedings of DNA 3*, Philadelphia, PA (H. Rubin, D. Wood, Eds.), *DIMACS* 48 (1997) pp. 57–72.
32. A. Hartemink, D. Gifford, Thermodynamic simulation of deoxyoligonucleotide hybridization for DNA computation, *Proceedings of DNA 3*, Philadelphia, PA (H. Rubin, D. Wood, Eds.), *DIMACS* 48 (1997) pp. 25–38.
33. A. Hartemink, D. Gifford, J. Khodor, Automated constraint-based nucleotide sequence selection for DNA computation, *Proc. of DNA 4*, Philadelphia, PA, (L. Kari, H. Rubin, D. Wood, Eds.), *Biosystems* 52 (1999) pp. 227–235.
34. J. Hopcroft, J. Ullman, R. Motwani, *Introduction to Automata Theory, Languages and Computation*, 2nd edition, Boston MA: Addison Wesley 2001.
35. S. Hussini, L. Kari, S. Konstantinidis, Coding properties of DNA languages, *Theoretical Computer Science* 290 (2003) 1557–1579.
36. N. Jonoska, K. Mahalingam, J. Chen, Involution codes: With application to DNA coded languages, *Natural Computing* 4 (2005) 141–162.
37. N. Jonoska, L. Kari, K. Mahalingam, Involution solid and join codes, *International Conference on Developments in Language Theory (DLT)*, Santa Barbara, CA, *LNCS* 4036 (2006) pp. 192–202.
38. L. Kari, *DNA* Computing: Arrival of biological mathematics, *The Mathematical Intelligencer* 19 (1997) 9–22.
39. L. Kari, R. Kitto, G. Gloor, A computer scientist's guide to molecular biology, *Soft Computing*, (G. Paun, T. Yokomori, Eds.), Vol. 5 (2001) pp. 95–101.
40. L. Kari, S. Konstantinidis, E. Losseva, G. Wozniak, Sticky-free and overhang-free DNA languages, *Acta Informatica* 40 (2003) 119–157.

41. L. Kari, S. Konstantinidis, E. Losseva, P. Sosik, G. Thierrin, Hairpin structures in DNA words, *Proceedings of DNA 11*, London, Ontario, Canada (A. Carbone, N. Pierce, Eds.), *LNCS* 3892 (2006) pp. 158–170.
42. L. Kari, K. Mahalingam, Watson-Crick conjugate and commutative words, *Proceedings of DNA 13*, Memphis, TN (M. Garzon, H. Yan, Eds.), *LNCS* 4848 (2008), pp. 273–283.
43. L. Kari, K. Mahalingam, Watson-Crick palindromes in DNA computing, *Natural Computing* (2009) DOI 10.1007/s11047-009-9131-2.
44. L. Kari, S. Konstantinidis, P. Sosik, Bond-free languages: Formalizations, maximality and construction methods, *International Journal of Foundations of Computer Science* 16 (2005) 1039–1070.
45. L. Kari, S. Seki, On pseudoknot-bordered words and their properties, *Journal of Computer and System Sciences*, 75 (2009) 113–121.
46. S. Kashiwamura, M. Yamamoto, A. Kameda, T. Shiba, A. Ohuchi, Potential for enlarging DNA memory: The validity of experimental operations of scaled-up nested primer molecular memory, *Biosystems* 80 (2005) 99–112.
47. J. Kim, J. Hopfield, E. Winfree, Neural network computation by in vitro transcriptional circuits, *Advances in Neural Information Processing Systems (NIPS)* 17 (2004) 681–688.
48. J. Kim, K. White, E. Winfree, Construction of an in vitro bistable circuit from synthetic transcriptional switches, *Molecular Systems Biology* 2 (2006) doi:10.1038/msb4100099.
49. T. Knight Jr., G. Sussman, Cellular gate technology, *Unconventional Models of Computation* (1998) 257–272.
50. L. Landweber, L. Kari, The evolution of cellular computing: nature's solution to a computational problem, *Proceedings of DNA 4*, Philadelphia, PA (L. Kari, H. Rubin, D. Wood, Eds.), *Biosystems* 52 (1999) 3–13.
51. X. Liang, H. Nishioka, N. Takenaka, H. Asanuma, Construction of photon-fuelled DNA nanomachines by tethering azobenzenes as engines, *Preliminary Proceedings of DNA 14*, Prague, Czech Republic (A. Goel, F. Simmel, P. Sosik, Eds.) June (2008) pp. 17–25.
52. T. Liedl, M. Olapinski, F. Simmel, A surface-bound DNA switch driven by a chemical oscillator, *Angewandte Chemie International Edition* 45 (2006) 5007–5010.
53. T. Liedl, T. Sobey, F. Simmel, DNA-based nanodevices, *Nanotoday* 2 (2007) 36–41.
54. R. Lipton, DNA solution of hard computational problems, *Science* 268 (1995) 542–545.
55. M. Lothaire, *Combinatorics on Words*, Cambridge University Press, Cambridge U. K. 1997.
56. C. Mao, T. LaBean, J. Reif, N. Seeman, Logical computation using algorithmic self-assembly of DNA triple-crossover molecules, *Nature* 407 (2000) 493–496.
57. A. Marathe, A. Condon, R. Corn, On combinatorial word design, *Journal of Computational Biology* 8 (2001) 201–219.
58. J. Mitchell, B. Yurke, DNA scissors, *Proceedings of DNA 7*, Tampa, FL (N. Jonoska, N. Seeman, Eds.), *LNCS* 2340 (2002), pp. 258–268.
59. H. Nakagawa, K. Sakamoto, Y. Sakakibara, Development of an *in vivo* computer based on *Escherichia Coli*, *Proceedings of DNA 11*, London, Ontario, Canada (A. Carbone, N. Pierce, Eds.), *LNCS* 3892 (2006) pp. 203–212.
60. A. Neel, M. Garzon, P. Penumatsa, Improving the quality of semantic retrieval in DNA-based memories with learning, *International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, Wellington, New Zealand, *LNCS* 3213 (2004) pp. 18–24.
61. V. Noireaux, A. Libchaber, A vesicle bioreactor as a step toward an artificial cell assembly, *Proceedings of the National Academy of Sciences* 101 (2004) 17669–17674.
62. V. Noireaux, R. Bar-Ziv, A. Libchaber, Principles of cell-free genetic circuit assembly, *Proceedings of the National Academy of Sciences* 100 (2003) 12672–12677.
63. M. Nowacki, V. Vijayan, Y. Zhou, K. Schotanus, T. Doak, L. Landweber, RNA-mediated epigenetic programming of a genome-rearrangement pathway, *Nature* 451 (2008) 153–158.

64. G. Paun, G. Rozenberg, A. Salomaa, *DNA Computing—New Computing Paradigms*, Springer-Verlag, Berlin, 1998.
65. R. Pei, S. Taylor, D. Stefanovic, S. Rudchenko, T. Mitchell, M. Stojanovic, Behaviour of polycatalytic assemblies in a substrate displaying matrix, *Journal of American Chemical Society* 128 (2006) 12693–12699.
66. D. Prescott, The DNA of ciliated protozoa, *Microbiology and Molecular Biology Reviews* 58 (1994) 233–267.
67. D. Prescott, A. Ehrenfeucht, G. Rozenberg, Template- guided recombination for IES elimination and unscrambling of genes in stichotrichous ciliates, *Journal of Theoretical Biology* 222 (2003) 323–330.
68. L. Qian, E. Winfree, A simple DNA gate motif for synthesizing large-scale circuits, *Preliminary Proceedings of DNA 14*, Prague, Czech Republic (A. Goel, F. Simmel, P. Sosik, Eds.), June (2008) pp. 139–151.
69. J. Reif, The design of autonomous DNA nanomechanical devices: Walking and rolling DNA, *Proceedings of DNA 8*, Sapporo, Japan (M. Hagiya, A. Ohuchi, Eds.), *LNCS* 2568 (2003) pp. 22–37.
70. J. Reif, T. LaBean, Autonomous programmable biomolecular devices using self-assembled DNA nanostructures, *Communications of the ACM* 50 (2007) 46–53.
71. J. Reif, T. LaBean, M. Pirrung, V. Rana, B. Guo, C. Kingsford, G. Wickham, Experimental construction of very large scale DNA databases with associative search capability, *Proceedings of DNA 7*, Tampa, FL (N. Jonoska, N. Seeman, Eds.), *LNCS* 2340 (2002) pp. 231–247.
72. P. Rothemund, Folding DNA to create nanoscale shapes and patterns, *Nature* 440 (2006) 297–302.
73. P. Rothemund, N. Papadakis, E. Winfree, Algorithmic self-assembly of DNA Sierpinski triangles, *PLoS Biology* 2 (2004) 2041–2053.
74. K. Sakamoto, H. Gouzu, K. Komiya, D. Kiga, S. Yokoyama, T. Yokomori, M. Hagiya, Molecular computation by DNA hairpin formation, *Science* 288 (2000) 1223–1226.
75. A. Salomaa, *Formal Languages*, Academic Press, New York, 1973.
76. G. Seelig, D. Soloveichik, D. Zhang, E. Winfree, Enzyme-free nucleic acid logic circuits, *Science* 314 (2006) 1585–1588.
77. N. Seeman, DNA in a material world, *Nature* 421 (2003) 427–431.
78. N. Seeman, Nanotechnology and the double-helix, *Scientific American* 290 (2004) 64–75.
79. H. Sekiguchi, K. Komiya, D. Kiga, M. Yamamura, A realization of DNA molecular machine that walks autonomously by using a restriction enzyme, *Proceedings of DNA 13*, Memphis, TN (M. Garzon, H. Yan, Eds.), *LNCS* 4848 (2008) pp. 34–65.
80. W. Sherman, N. Seeman, A precisely controlled DNA biped walking device, *Nano Letters* 4 (2004) 1203–1207.
81. W. Shih, J. Quispe, G. Joyce, A 1.7-kilobase single-stranded DNA that folds into a nanoscale octahedron, *Nature* 427 (2004) 618–621.
82. J. Shin, N. Pierce, A synthetic DNA walker for molecular transport, *Journal of American Chemical Society* 126 (2004) 10834–10835.
83. H. J.Shyr, *Free Monoids and Languages*, Hon Min Book Company, Taichung, Taiwan, 2001.
84. F. Simmel, B. Yurke, Using DNA to construct and power a nanoactuator, *Physical Review* E63 (2001) 041913.
85. G. Smith, C. Fiddes, J. Hawkins, J. Cox, Some possible codes for encrypting data in DNA, *Biotechnology Letters* 25 (2003) 1125–1130.
86. D. Soloveichik, E. Winfree, The computational power of Benenson automata, *Theoretical Computer Science* 344 (2005) 279–297.
87. Y. Tian, Y. He, Y. Chen, P. Yin, C. Mao, A DNAzyme that walks processively and autonomously along a one-dimensional track, *Angewandte Chemie International Edition* 44 (2005) 4355–4358.
88. Y. Tsuboi, Z. Ibrahim, O. Ono, DNA-based semantic memory with linear strands, *International Journal of Innovative Computing, Information and Control* 1 (2005) 755–766.

89. D. Tulpan, H. Hoos, A. Condon, Stochastic local search algorithms for DNA word design, *Proceedings of DNA 8*, Sapporo, Japan (M. Hagiya, A. Ohuchi, Eds.), *LNCS* 2568 (2003) pp. 229–241.
90. H. Wang, Proving theorems by pattern recognition—II, *Bell System Technical Journal* 40 (1961) 1–41.
91. J. Watson, N. Hopkins, J. Roberts, J. Steitz, A. Weiner, *Molecular Biology of the Gene*, Benjamin Cummings Menlo Park, CA, 1987.
92. R. Weiss, G. Homsy, T. Knight, Jr., Toward in-vivo digital circuits, *Evolution as Computation, Natural Computing Series*, Springer, Berlin, Heidelberg, New York (2002) 275–295.
93. R. Weiss, S. Basu, The device physics of cellular logic gates, *Proceedings of the First Workshop on Non-Silicon Computation*, Cambridge, MA (2002) 54–61.
94. E. Winfree, F. Liu, L. Wenzler, N. Seeman, Design and self-assembly of two-dimensional DNA crystals, *Nature* 394 (1998) 539–544.
95. E. Winfree, X. Yang, N. Seeman, Universal computation via self-assembly of DNA: some theory and experiments, *Proceedings of DNA 2*, Princeton, NJ (L. Landweber, E. Baum, Eds.), *DIMACS* 44 (1996) 191–213.
96. E. Winfree, Whiplash PCR for O(1) computing, Caltech Technical Report, CaltechCSTR:1998.23 (1998).
97. P. Wong, K. Wong, H. Foote, Organic data memory using the DNA approach, *Communications of the ACM* 46 (2003) 95–98.
98. N. Yachie, K. Sekiyama, J. Sugahara, Y. Ohashi, M. Tomita, Alignment based approach for durable data storage into living organisms, *Biotechnology Progress* 23 (2007) 501–505.
99. M. Yamamoto, S. Kashiwamura, A. Ohuchi, DNA memory with 16.8M addresses, *Proceedings of DNA 13*, Memphis, TN (M. Garzon, H. Yan, Eds.), *LNCS* 4848 (2008) pp. 99–108.
100. X. Yang, A. Vologodskii, B. Liu, B. Kemper, N. Seeman, Torsional control of double stranded DNA branch migration, *Biopolymers* 45 (1998) 69–83.
101. T. Yokomori, Yet another computation model of self-assembly, *Proceedings of DNA 5*, Cambridge, MA (E. Winfree, D. Gifford, Eds.), *DIMACS Series* 54 (1999) 153–167.
102. S. S. Yu, *Languages and Codes*, Lecture Notes, Department of Computer Science, National Chung-Hsing University, Taichung, Taiwan, 2005.
103. B. Yurke, A. Turberfield, A. Mills, F. Simmel, J. Neumann, A DNA-fuelled molecular machine made of DNA, *Nature* 406 (2000) 605–608.