Chapter 8: Nanocomputing

Jennifer Sager[1], Joseph Farfel[2], and Darko Stefanovic[1]
[1]Department of Computer Science, University of New Mexico
[2]Department of Computer Science, Duke University

Contact author:
Darko Stefanovic
Department of Computer Science
University of New Mexico
MSC01 1130
1 University of New Mexico
Albuquerque, NM 87131
USA
phone: +1 505 2776561
fax: +1 505 2776927
darko@cs.unm.edu

July 12, 2006

Chapter 8

Nanocomputing

Jennifer Sager[1], Joseph Farfel[2], and Darko Stefanovic[1]

[1]Department of Computer Science, University of New Mexico

[2]Department of Computer Science, Duke University

Summary. Nanocomputing encompasses any submicron devices and technologies applied to any computational or related tasks. A brief survey is given, and emphasis is placed on biomolecular devices that use nucleic acids as their substrate. Computational self-assembly of DNA, and DNA-based enzymatic computing are surveyed in greater detail. The foremost implementation challenge for computation, viz., DNA word design, is also surveyed.

Keywords. *Nanocomputing; DNA computing; DNA self-assembly; enzymatic computing; universal computation; DNA word design.*

Computing as we know it is based on the von Neumann stored program concept and its ubiquitous implementation in the form of electronic instruction processors. For the past three decades, processors have been fabricated using semiconductor integrated circuits, the dominant material being silicon, and the dominant technology CMOS. Relentless miniaturization has been decreasing feature size and increasing both the operating frequency and the number of elements per chip, giving rise to so-called Moore's law. Indeed, vast amounts of raw computational power are now available in every personal computer sold, at a very modest cost. By improving the processes and materials and using new geometries, the semiconductor industry expects to be able to continue this trend for at least another decade, according to its common Roadmap document (1). Whereas a 90 nm node is characteristic of current processes (implying that the semiconductor industry is already operating in the nanotechnology domain), it is expected that 18 nm will be reached by 2018. Beyond that lie fundamental limits of the technology, principally the problem of heat dissipation (2,3) inherent to devices in which an electronic charge is used for state representation. Alternatives are being sought to CMOS fabrication (4–8) at the level of devices, such as single-electron transistors (9, 10), carbon nanotubes (11), silicon nanowires (12–14), molecular switches (15–17), nanomagnets (18), quantum dots (19), chemically assembled electronics (20–29), chemical logic gates with optical outputs (30–34), and three-dimensional semiconductor integration (35) (pre-

dicted much earlier (36)). Alternative architectures are also being explored, such as amorphous computing (37), spatial computing (26, 38) blob computing (39, 40), cell matrix computing (41), chaos computing (42), and the entire field of quantum computing.

Thus, while we need not fear a scarcity of computing cycles, the prospect of eventual demise of Moore's law has given impetus to a great variety of research into new computational substrates. A separate chapter in this volume treats nanoelectronics, that is, work that aims to, more or less seamlessly, extend the viability of microelectronic technologies beyond the lifetime of CMOS processes. Here we focus on research over the past decade that has been less concerned with continuity, and that attempts to achieve computational effects through the application of biochemical principles in new and unexpected ways. Our main focus is on various computing paradigms using DNA. We examine in which sense they perform computation and interpret them in terms of conventional mathematical notions of computation. We also examine their commonalities, in particular the question of DNA word design.

DNA computation in its original formulation (43–49) seeks to employ the massive parallelism inherent in the small scale of molecules to speed up decision problems. The essential property of nucleic acids, specific hybridization (formation of the double helix) (50–53) is either exploited to encode solutions as long strings of nucleotides, generate large numbers of random strings and check them in a small number of steps, often manual such as PCR (though more reliable detection is now available (54)), or to construct solutions directly through oligonucleotide self-assembly. A number of NP-complete decision problems have been rendered in this fashion (55–58), and encodings for general computation (59–63) and combinatorial games (64) have also been proposed. A limitation of the approach is the need for large amounts of nucleic acid (65); with amounts currently feasible (and the low speed of operations), it has been difficult to outperform electronic computers. Another limitation has been in imperfect specificity of nucleic acid hybridization. The research in this area (66–73) has ranged from the physico-chemical constraints on usable nucleotide strings (e.g., melting points; secondary structure) to tools for systematic string generation (74); we review it in Section 8.3.

Further variations on the theme of DNA computation have included using proteins instead of nucleic acids, for a larger alphabet (75), hairpin computation (76), sophisticated forms of self-assembly (77), to avoid manual operations, and cellular computation in which cells (real or simulated) are viewed as elementary computational elements, with some form of communication among multiple cells (51, 77–93).

While early on it was believed that DNA computing might be a competitor to electronics in solving hard computational problems, the focus has now shifted to the use of DNA to compute in environments where it is uniquely capable of operating, such as in smart drug delivery to individual cells (94, 95).

Our review of biochemically based computing, necessarily limited in scope, is organized according to the manner in which the principle of specific hybridization is exploited. In Section 8.1 we consider how large two- and three-dimensional structures are built in a programmable fashion through molecular self-assembly. In Section 8.2 we treat approaches in which short strands representing logic signals spcifically bind to activate particular enzymatic reactions in a reaction network. Finally, the pervasive subproblem of the design of good DNA sequences for computation

is treated in Section 8.3.

8.1 Computing using structural self-assembly of DNA

One of the most interesting and useful paradigms in biomolecular computation is molecular self-assembly. *Self-assembly* is the spontaneous formation of ordered structure out of structural building blocks which encode within themselves information about both what they are and how they can fit together. Useful computation can occur if the rules that govern how certain types of blocks may attach to other types of blocks are intelligently selected.

In the molecular case, the building blocks which self-assemble are normally DNA molecules. DNA is perfect for self-assembly because pieces of DNA may be linked together in very programmable and predictable ways. In fact, we can construct many different building block structures with DNA, and we can program how these blocks attach to each other to achieve infinitely variable superstructures—indeed, DNA self-assembly has even been proven to be capable of universal computation.

8.1.1 Building Blocks

The most familiar form of DNA is the double-stranded, or *dsDNA* molecule. These molecules consist of two backbones which wrap around each other in a double-helix (50), and are connected by Watson-Crick complementary bonds between the amino acids A, C, T, and G (adenine, cytosine, thymine, and guanine). Watson-Crick complementary bonding refers to the fact that these four amino acids form two pairs of acids which bind very strongly to each other—A binds to T, and C binds to G.

Double-stranded DNA, or dsDNA, may be used as a building block for self-assembly. In order for pieces of dsDNA to self-assemble, though, they need to have outreaches of single-stranded DNA at their ends. We call these extending segments *sticky ends*, because a segment of single-stranded DNA will bind (stick) to another segment of single-stranded DNA, or ssDNA, that contains a sequence of amino acids which is Watson-Crick complementary to its own sequence. If multiple pieces of dsDNA have sticky ends on both sides, they can link together to form a long chain. The initial bonding of the amino acids of one piece of ssDNA to another is called *hybridization*. After hybridization, the pieces may complete their attachment through a process called *ligation*, where the DNA backbone is extended and connected. See Figure 8.1 for an illustration of these reactions between pieces of double-stranded DNA with extending sticky ends (96).

Pieces of dsDNA are linear, and therefore are inadequate building blocks for the construction of any two- or three-dimensional structures. This has led researchers to use other types of DNA molecules, beyond the standard double-helix, for producing complex structures. The first type of molecules are called junction molecules.

Junction molecules are formed when two strands of dsDNA undergo reciprocal exchange (recombination), whereby they fuse together at what is called a branched junction, or Holliday junction (see Figure 8.2) (97). In reciprocal exchange, the strands of DNA fuse by exchanging connections at a particular site. This may happen between dsDNA molecules of the same or opposite

polarity, and although either polarity combination yields the same structure after one crossover, different structures are achieved if more exchanges occur (molecules are of the same polarity if they are arranged such that the two strands which undergo reciprocal exchange have the same orientation of their 3' and 5' ends). A junction molecule may be constructed with an arbitrary number of arms, and there is no known limit to this number (97). Figure 8.2 shows a five-arm junction made from a four-arm junction and a hairpin DNA molecule. We may link junction molecules together to form more complicated structures if we extend a bit of single-stranded DNA off each arm of a junction molecule, creating sticky ends on the arms. Molecules with topologies resembling the edges of a cube and a truncated octahedron have been demonstrated (97). However, structures made out of singly branched junctions are relatively flexible, and so it is impossible to characterize the actual three-dimensional structure of these molecules. To create predictable complex structures from DNA molecules, more rigidity is needed than that provided by branched junctions. Another class of molecules called DNA crossovers offers this rigidity.

A *DNA crossover molecule* is a structure consisting of two dsDNA molecules, where each dsDNA molecule has a single strand that crosses over to the other molecule (see Figure 8.3) (96). This is just reciprocal exchange between the two molecules happening at multiple sites. The two most significant types of crossover molecules are *double crossovers*, or *DX* molecules, and *triple crossovers*, or *TX* molecules. DX molecules are made up of two pieces of dsDNA, with two crossover locations (93). TX molecules are made up of three pieces of dsDNA with four crossovers (62). We may extend sticky ends off DX and TX molecules to link them together, and call the linkable molecules *tiles*, in the manner of Wang tiles, which are discussed in the next section. These DX and TX tiles are sufficiently rigid to create very complex, stable, and beautiful two- and three- dimensional nanostructures via self-assembly, and, with intelligent selection of how different pieces may attach, this assembly may also be used to perform computation.

8.1.2 Computation

Erik Winfree was the first to discover that planar self-assembly of DNA molecules can perform universal computation (48). This discovery was made based on the insight that DX molecules may be regarded simply as *Wang tiles*. Wang tiling is a mathematical model where square unit tiles are labeled with specific symbols on each edge. Each tile is only allowed to associate with tiles that have matching symbols. We can construct DNA molecules that are analogous to Wang tiles (call these DNA tiles) by creating a molecule with a rigid, stable body and open, sticky ends for attachment to other tiles. The DX and TX molecules are both ideal for this. The sticky ends of DNA tiles may be labeled with certain sequences of amino acids, which are analogous to the symbols labeling the sides of Wang tiles. This labeling allows the sticky ends to bind only to tile ends that have a complementary sequence of base pairs; this corresponds to the rule that restricts Wang tiles to only associate with tiles that have matching symbols. It has been shown that Wang tiles, when designed with a certain set of symbols, are capable of universal computation, and since DNA molecules can represent Wang tiles, it was shown that universal computation could also be accomplished by self-assembling DNA tiles (93, 98).

The biggest advantage of computing with self-assembly, compared to other molecular computing paradigms, is that it avoids the many tedious laboratory steps that are requirements of other

computation methods. The reason for this is that if DNA tiles are designed to correctly specify the desired steps in a computational problem, the only structures to form from these tiles will be the desired, valid solutions of the problem. Since only valid solutions are encoded in the resulting structures, one needs only to design and form the tiles from DNA strands, allow the tiles to self-assemble, and then read the output. Of course, reading the output usually involves at least two main steps, such as ligation of reporter strands embedded in the tiles, and subsequent separation and PCR. However, the number of total steps in performing computation with self-assembly remains very low.

The first example of computing performed by DNA self-assembly was a four-bit cumulative XOR (62). The function XOR takes two binary input bits and returns a zero if the inputs are equal and a one if they are not equal. The cumulative XOR takes Boolean input bits $x_1, \ldots, x_n$, and computes the Boolean outputs $y_1, \ldots, y_n$, where $y_1 = x_1$, and for $i > 1$, $y_i = y_{i-1}\text{XOR}x_i$. The effect of this is that y_i is equal to the even or odd parity of the first i values of x. The cumulative XOR calculation was performed via the self-assembly of triple-crossover, or TX molecules. Eight types of TX molecule were needed: two corner tiles, two input tiles, and four output tiles. The types were different only in that they had different labels (sequences of amino acids) on their sticky ends, and, in some cases, different numbers of sticky ends. The corner tiles were used to to connect a layer of input tiles to a layer of output tiles. The two input tiles represented $x_i = 0$ and $x_i = 1$. The four output tiles were needed because there are two ways to get each of the two possible outputs of a bitwise XOR. So, one output tile represents the state where we have output bit $y_i = 0$ and input bits $x_i = 0$ and $y_{i-1} = 0$, while another tile represents the state where we have output bit $y_i = 0$ and input bits $x_i = 1$ and $y_{i-1} = 1$. Similarly, the other two output tiles represent the two states where $y_i = 1$. The actual computation of the XOR operation is accomplished by harnessing the way the output tiles connect to the input tiles. Each output tile (y_i) will only attach to a unique combination of one input tile (x_i) and one output tile (y_{i-1}), and will leave one sticky end open that represents its own value (y_i) so that another output tile may attach to it. For example, the output tile signifying $y_i = 1$, $x_i = 0$, and $y_{i-1} = 1$ has the value 1, and will only connect to an input tile with value 0 and an output tile with value 1. With this system, only the output tiles that represent the correct solution to the problem will be able to attach to the input tiles.

Another example of computation using self-assembled DNA tiles is the binary counter created by Rothemund and Winfree (63). The counter uses seven different types of tiles: two types of tiles representing 1, two types representing 0, and three types for the creation of a border (corner, bottom, and side tiles). The counter works by first setting up a tile border with the border tiles—it is convenient to think of the "side" border tiles to be on the right, as then the counter will read numbers from left to right. The border structure forms before the rest of the counter because of the properties of border tiles: Two border tiles bind together with a double bond, while all other tiles bind to each other and to border tiles with a single bond. Doubly-bound tiles have a very low tendency to detach from each other, while singly-bound tiles detach relatively easily. Since any tile except a border tile must bind to two additional tiles in order to have two bonds, but a border tile and another border tile of the correct type will form a double-bond with each other, a stable border forms before other stable formations, composed of non-border tiles, are created. The bottom and side border tiles are designed such that the only tile that may bind in the border's corner (to both

a side and a bottom border tile) is a specific type of 1 tile. Only one of the 0 tiles may bind to both this 1 tile and the bottom of the border, and this type of 0 tile may also bind to itself and the bottom of the border, and thus may fill out the left side of the first number in the counter with leading zeros. Now, the only type of tile which may bind both above the 1 in the corner and to the right side of the border is the other type of 0 tile, and the only tile which may bind to the left of it is a 1 tile—we get the number 10, or two in binary. The tile binding rules are such that this can continue similarly up the structure, building numbers that always increment by one. Figure 8.4 shows a more intuitive picture of this device's operation.

DNA self-assembly has also been used to solve the Boolean formula satisfiability (SAT) problem. This has been done with both string (linear) assembly of DX or TX tiles and with graph self-assembly of duplex and branched junction molecules (99). In the string assembly solution, the DNA tiles have a width (the number of helixes that are fused together) equal to the number of clauses in the SAT problem. Each variable involved in the problem has two tiles, one representing its being true, and one representing its being false. A variable's "true" tile has a hairpin structure in each clause where the variable appears, and no hairpin in clauses where its complement appears (where the variable is false). The same applies for a variable's "false" tile. When all the tiles are mixed together (including a "start" and an "end" tile), they all join together to form only valid solutions of the SAT problem.

8.1.3 Complex Nanostructures

In addition to performing computation, DNA tiles can self-assemble to create very complex 2D and 3D geometrical structures. Two-dimensional periodic lattices have been constructed of both double-crossover (DX) and triple-crossover (TX) DNA tiles (51). Both types of lattice have been observed through atomic force microscopy, to see that the desired geometric structure is actually being self-assembled. To assist the visualization of the structure, a lattice made of TX molecules may be designed in such a way that rows of molecules contain loops of DNA that protrude perpendicularly to the plane of the lattice. These rows can be placed at regular distances that can be designated with high accuracy (in the lab, stripes were seen at 27.2 nm when they were expected at 28.6 nm) (51). The stripes can be seen even more clearly when metallic (normally gold) balls are affixed to the tiles making up the stripes (96).

Recently, researchers have proposed methods of making complex nanoscale three-dimensional fractals. Specifically, a method has been proposed by which the Sierpinski cube fractal could be produced using DNA self-assembly (100). The recursive algorithm for generating a Sierpinski cube fractal is as follows: take a cube, divide it evenly into 27 smaller cubes, and remove the most interior cube as well as the middle cubes on the large cube's 6 faces. Research has shown theoretically that the cube may be produced by using Mao triangles based on DX molecules. However, the cube has not yet been produced in the lab.

8.1.4 Errors and Error Correction

Atomic force microscopy has allowed us to view self-assembled DNA structures and investigate whether or not they are forming properly. There is indeed great success, but this has also allowed us

to see that there are problems with reliably building large, error-free structures. The self-assembled binary counter, for example, is error-prone in its current incarnation, only counting to 7 or 8 accurately (101). In fact, there is a 1 to 10 percent error rate for each tile binding in all two-dimensional structures constructed without any error correction or error avoidance techniques (102). This can lead to disastrous results in many computations; such error rates come with the new territory of biological computation, and are not a problem that traditional computer scientists are at all accustomed to dealing with.

There are three main kinds of assembly error (103). See Figure 8.5 for visual examples of each. The first kind of error is a *mismatch error*, where sometimes tiles become locked in the assembly in the wrong place. A tile can attach to a corner in the assembly's fringe by binding to one tile at the corner, but mismatching with the other. Normally a tile in such a state would fall off the assembly, since two bonds (or one strong bond, as is the case with border tiles) are necessary for a tile to be locked in a stable position. However, if other tiles attach around it before it falls off, it may be bound to enough tiles to be locked in a stable, but incorrect position in the assembly. It is easy to see that just one tile locked in an incorrect position will throw the binary counter completely off course, as the assembly of each row of digits in the counter is dependent upon the previous row. While some other self-assembled patterns may be less sensitive, the fact remains that even one erroneously placed tile can greatly impact the structure of an assembly.

The second kind of error is a *facet error*. This happens when a tile attaches to a facet (a portion of the boundary apart from the built interior structure) rather than to a desired attachment site at a corner in the structure's frontier, and more tiles bind it into place. Even though no mismatches occur, an incorrect structure can be formed this way.

The third kind of error is a *spurious nucleation error*. This occurs when the assembly begins growing from a tile other than the special "seed" tile (normally the corner of the lattice). For example, a portion of the interior can spontaneously come together without any boundary tiles at all. More commonly, though, a stretch of boundary tiles will bind together without being bound to the seed tile. A section of linked boundary tiles floating around without a seed tile to set up the assembly structure is a perfect recipe for facet errors, since the seed tile, which links two boundary lines together, is necessary to create the first desired binding site for the main body of the lattice, in the corner where the boundaries meet. Any binding of tiles to a boundary line not linked to a seed tile constitutes facet error. Avoiding spurious nucleation when running a self-assembly algorithm is analogous to providing correct inputs to the beginning of a computer program; in other words, growing from the seed tile makes the algorithm begin with the input you want.

All these errors occur because of the reversible, kinetic way in which DNA molecules in solution react and bind together. While two bonds (or one double-bond) are indeed required to hold a tile in a stable spot in an assembly, in reality there are many times where a tile will attach to the assembly with only one bond, and hang on for a little while, sometimes allowing itself to be locked into place with further bonds. Likewise, it is also possible that even the strong double bonds may be reversed, and break apart, at times. A kinetic Tile Assembly Model (kTAM) has been created (by Winfree and others) to simulate reversible tile interactions. The kTAM approximates perfect, abstract self-assembly with strength threshold τ (given as a property of the tile program) when $G_m = \tau G_s - \varepsilon$, with G_m being the monomer tile concentration and G_s being the sticky-end bond

strength; ε is the error rate. The model defines the forward rate of crystal growth (association) of particular tiles as $r_f = k_f e^{-G_m}$, where k_f is a reaction constant. The backward rate of growth (dissociation) of a tile which makes bonds with total strength b is $r_{r,b} = k_f e^{-bG_s}$. The free energy of a nucleus of tiles is defined as $\Delta G = (bG_s - nG_m)kT$, where b is the total bond strength, n is the number of tiles, k is Boltzmann's constant, and T is temperature. These measures help determine under what conditions assembly steps are energetically favorable (and thus have higher probability of occurring at any given point in time).

Perhaps obviously, we can account for most errors just by slowing down the rate at which structures assemble. Research has shown, however, that mismatch errors occur at a rate which is at least proportional to the square root of the speed of assembly (77). Thus, in order to reduce the rate of error by some reasonable amount, we must slow the rate of assembly down tremendously, by greatly decreasing the temperature and/or the monomer concentration. Better solutions are being investigated, then, for lowering error rates.

The most promising methods involve using *proofreading tiles* (104–106). These methods can greatly help in controlling both mismatch and facet errors. Proofreading tiles are extra tiles added to a tile set that are used to store information redundantly, so it is harder to lock errors in place in a forming structure. This type of error correction forces errors to be co-localized, so that many more erroneous tile bindings must occur before one wrong tile is locked in place. This greatly increases the probability that an individual wrong tile will fall off the assembly before growth continues around it, thus substantially reducing the error rate in building the assembly. Each tile is replaced by a block of tiles, where the bind between each pair of tiles in the block is unique (105). When using a simple 2x2 array of proofreading tiles, the tile set for a given problem is four times larger in size, but the error rate is 10^4 lower (104). Originally, the internal binding between proofreading tiles was very simple, but Chen and Goel have improved upon this to produce the "snake" proofreading method. A snake tile set forces the assembly process to double, or "snake" back onto itself when binding each proofreading block, making it less likely that an entire block will be bound incorrectly to the growing structure (104). With either type of proofreading tile set, the mismatch and facet error rates can be made arbitrarily small by using larger and larger tile sets (although this produces larger and more redundant self-assembled lattices, of course). See Figure 8.6 for an example of both types of proofreading tile sets.

The "zig-zag" boundary tile set helps prevent spurious nucleation errors, by forcing border tiles to bind to seed tiles before binding to each other (103). This border tile set makes it more energetically favorable for border tiles to bind correctly, so a complete border structure (with seed in place) is set up before the rest of the structure begins growing. The zig-zag border construction method can be combined with the proofreading tile sets mentioned earlier to yield a self-assembled creation that is robust to all three types of error.

8.2 Enzymatic DNA computing

In this section we focus on the approach to biochemical computing—either digital or analog, depending on the interpretation—in which signals are represented by concentrations of designated molecular species. While such systems can be devised with protein enzymes, here we look at

smaller DNA enzyme molecules. Deoxyribozymes are enzymes made of DNA that catalyze DNA reactions such as by cleaving a DNA strand into two or ligating two strands into one. Cleaving enzymes (known as phosphidiesterases) can be modified to include allosteric regulation sites to which specific control molecules can bind and so affect the catalytic activity. There is a type of regulation site to which a control molecule must bind before the enzyme can complex with (i.e., bind to) the substrate, thus the control molecule promotes catalytic activity. Another type of regulation site allows the control molecule to alter the conformation of the enzyme's catalytic core, such that even if the substrate has bound to the enzyme, no cleavage occurs; thus this control molecule suppresses or inhibits catalytic activity. This allosterically regulated enzyme can be interpreted as a logic gate, the control molecules as inputs to the gate, and the cleavage products as the outputs. This basic logic gate corresponds to a conjunction, such as e.g., $a \wedge b \wedge \neg c$, here assuming two promotory sites and one inhibitory site, and using a and b as signals encoded by the promotor input molecules and c as a signal encoded by the inhibitor input molecule. Deoxyribozyme logic gates are constructed via a modular design that combines molecular beacon stem-loops (107) with hammerhead-type deoxyribozymes, Figures 8.7.

A gate is active when its catalytic core is intact (not modified by an inhibitory input) and its substrate recognition region is free (owing to the promotive inputs), allowing the substrate to bind and be cleaved. Correct functioning of individual gates can be experimentally verified through fluorescent readouts **F** (108).

Note that the gates use oligonucleotides as both inputs and outputs, so cascading gates is possible without any external interfaces (such as e.g., photoelectronics). The inputs are compatible with sensor molecules (109) that could detect cellular disease markers. Final outputs can be tied to release of small molecules. Two gates are coupled *in series* if the product of an "upstream" gate specifically activates a "downstream" gate. All products and inputs (i.e., external signals) must be sufficiently different to minimize the error rates of imperfect oligonucleotide matching, and they must not bond to one another; we examine this problem in the next section. A series connection of two gates, the upstream being a ligase and the downstream being a phosphodiesterase, has been experimentally validated (110).

Multiple elementary gates have been constructed, so there is a large number of equivalent ways that any given Boolean function can be realized—equivalent in terms of digital function, but not in speed or cost of realization. For instance, a single four-input gate may be preferable to a cascade with three two-input gates. Clearly construction of deoxyribozyme logic circuits bears resemblance to traditional low-level logic design, but, perhaps because the technology has not matured, with many more options to explore.

8.2.1 Simple enzymatic circuits

Deoxyribozyme logic gates have been used to build computational devices. A half-adder was achieved by combining three two-input gates in solution (111). A half-adder computes the sum of two binary digits (bits); there may be a carry. It can be implemented using an XOR gate for the sum bit and an AND gate for the carry bit. The XOR gate, in turn, is implemented using two ANDNOT gates. The two substrates used are fluorogenically marked, red tetramethylrhodamine (T), green fluorescein (F), and the activity of the device can be followed by tracking the fluorescence at two

distinct wavelengths. The results, in the presence of Zn^{2+} ions, are shown in Figure 8.8. When both inputs are present, only the green fluorescein channel (carry bit) shows a rise in fluorescence. When only input i_1 is present or only input i_2 is present, only the red tetramethylrhodamine channel (sum bit) rises. With no inputs, neither channel rises. Thus, the two bits of output can be reliably detected and are correctly computed.

8.2.2 Enzymatic game automata

Using deoxyribizyme logic gates, an automaton for the game of tic-tac-toe has been constructed (112). To understand how this was achieved, we first briefly examine the structure of that game. A *sequential game* is a game in which players take turns making decisions known as *moves*. A *game of perfect information* is a sequential game in which all the players are informed before every move of the complete state of the game. A *strategy* for a player in a game of perfect information is a plan that dictates what moves that player will make in every possible game state. A *strategy tree* is a (directed, acyclic) graph representation of a strategy. The nodes of the graph represent reachable game states. The edges of the graph represent the opponent's moves. The target node of the edge contains the strategy's response to the move encoded on the edge. A leaf represents a final game state, and can, usually, be labelled either win, lose, or draw. Thus, a path from the root of a strategy tree to one of its leaves represents a game.

In a tree, there is only one path from the root of the tree to each node. This path defines a set of moves made by the players in the game. A player's *move set* at any node is the set of moves made by that player up to that point in a game. For example, a strategy's move set at any node is the set of moves dictated by the strategy along the path from the root to that node. A strategy is said to be *feasible* if, for every pair of nodes in the decision tree for which the opponent's move sets are equal, one of the following two conditions holds: (1)the vertices encode the same decision (i.e., they dictate the same move), or (2) the strategy's move sets are equal. A feasible strategy can be successfully converted into Boolean logic implemented using monotone logic gates, such as the deoxyribozyme logic gates.

In the tic-tac-toe automaton, the following simplifying assumptions. are made to reduce the number and complexity of needed molecular species. The automaton moves first and its first move is into the center (square 5, Figure 8.9). Because of symmetry, the first move of the human, which must be either a side move or a corner move, is restricted to be either square 1 (corner) or square 4 (side).

The game tree in Figure 8.10 represents the chosen strategy for the automaton. For example, if the human opponent moves into square 1 following the automaton's opening move into square 5, the automaton responds by moving into square 4 (as indicated on edge 21). If the human then moves into square 6, the automaton responds by moving into square 3 (edge 22). If the human then moves into square 7, the automaton responds by moving into square 2 (edge 23). Finally, if the human then moves into square 8, the automaton responds by moving into square 9, and the game ends in a draw.

This strategy is feasible; therefore, following a conversion procedure, it is possible to reach a set of Boolean formulae that realize it, given in Table 8.1. (For a detailed analysis of feasibility conditions for the mapping of games of strategy to Boolean formulae, see (113).) The arrangement

of deoxyribozyme logic gates corresponding to the above formulae is given in Figure 8.11. This is the initial state of the nine wells of a well-plate in which the automaton is realized in the laboratory.

The play begins when Mg^{2+} ions are added to all nine wells, activating only the deoxyribozyme in well 5, i.e., the automaton to play its first move into the center. After that, the game branches according to the opponent's inputs. A representative game is shown in Figure 8.12. As the human opponent adds input to indicate his moves, the automaton responds with its own move, activating precisely one well, which is shown enlarged. The newly activated gate is shown in light green. The bar chart shows the measured change in fluorescence in all the wells. Wells that are logically inactive (contain no active gates) have black bars, and wells that are logically active have green bars (the newly active well is light green).

8.2.3 Open systems and recurrent circuits

The first oscillatory chemical reaction was discovered by Belousov in the fifties but for a while remained little known (114). Once this Belousov-Zhabotinsky reaction became better known and its mechanisms understood (115–117), it inspired treatments of chemical computation devices, made out of hypothetical large systems of coupled chemical reactions with many stable states (118–126); moreover information-theoretic connections were made with Maxwell's daemon (127), and, chaotic behavior having been observed, with unpredictability (128–130). Chemical reactions, owing to diffusion, have a spatial component in addition to the temporal. Therefore the oscillatory Belousov-Zhabotinsky reaction gives rise to waves (131); this was used to implement computation on a prefabricated spatial pattern by wave superposition (132–134). Recently an oligonucleotide periodic system was shown (135) (see also (136)).

It has been suggested that computational devices based on chemical kinetics are Turing-equivalent (137), but one must consider the inherently finite number of reactions and molecular species possible (138), and the difficulty of constructing them in practice, beyond *Gedankenmoleküle* such as those of Hiratsuka (139). Deoxyribozyme logic provides a systematic method for such a construction, and recurrent circuits, including flip-flops and oscillators, have been designed *in silico* on the basis of it (140, 141).

8.3 Word design for DNA computing

Most DNA computation models assume that computation is error-free. (Even though we describe most of the constraints in terms of DNA, RNA computers also exist (for an example see (64)) and all of the constraints described here are also relevant to RNA.) For example, Adleman (43) and Lipton (45) used randomly generated DNA strings in their experiments because they assumed that errors due to false positives were rare. However, it has been experimentally shown that randomly generated codes are inadequate for accurate DNA computation as the size of the problem grows (68), since a poorly chosen set of DNA strands can cause hybridization errors. Therefore, for many types of DNA computers, it may be practical or even necessary to create a 'library' or 'pool' of DNA word codes suitable for computation.

There are three steps to constructing a library. First, rules or constraints must be defined which specify whether a given set of molecules will cause errors; these constraints can be complex since they are subject to the laws of biochemistry as well as the specific algorithm and computation style. Second, an algorithm must be found which either generates or finds such a set of molecules; the solution space is large because the number of candidate molecules grows exponentially in the length of the DNA string. Third, it must be proved that the final set of molecules correctly implements the DNA algorithm; for some problem instances proving this is NP-hard (142). Correspondingly, we define three problems in library design. Given an algorithm for a type of DNA computer, the DNA Code *Constraint* Problem is to find a set of constraints that the DNA strands must satisfy to minimize the number of errors due to the choice of DNA strands. Given a set of constraints, the DNA Code *Design* Problem is to find the largest set of DNA strands which satisfy the given constraints or to find a set of DNA strands of a given size that satisfy a given set of contraints the best. The DNA Code *Evaluation* Problem is to evaluate how accurate a set of DNA strands is for implementing a DNA algorithm.

8.3.1 DNA Code Constraint Problem

A properly constructed library will help to minimize errors so that DNA computation is more practical, reliable, scalable, and less costly in terms of materials and laboratory time. (For an overview of library design see (67). For a survey of algorithms that have been used to solve the DNA/RNA Code Design Problem see (143).) However, the construction of a library is non-trivial for two reasons. First, there are 4^N unique DNA strings of length N; thus the number of candidate molecules grows exponentially in the length of the DNA string. Second, the constraints used to find a library are complex since they are subject to the laws of biochemistry as well as the specific algorithm and computation style.

Positive And Negative Design

Even though there are many types of DNA computers, most share similar biochemical requirements because they use the same fundamental biochemical processes for computation. The fundamental computation step for most DNA computers occurs through the bonding (hybridization) and unbonding (denaturation) of oligonucleotides (short strands of DNA).

Creating an error-free library typically requires that planned hybridizations and denaturations (between a word and its Watson-Crick complement) do occur and unplanned hybridizations and denaturations (between all other combinations of code words and their complements) do not occur. The former situation is referred to as the *positive design problem* while the latter is referred to as the *negative design problem* (143, 144).

The positive design problem requires that there exists a sequence of reactions that produces the desired outputs, starting from the given inputs. Thus, positive design attempts to "optimize affinity for the target structure" (144). These reactions must occur within a reasonable amount of time for feasible concentrations. Usually the strands must satisfy a specified secondary structure criterion (e.g., the strand must have a desired secondary structure or have no secondary structure at all). Since a strand is typically identified by hybridization with its perfect Watson-Crick complement,

the positive design problem requires that each Watson-Crick duplex is stable. In addition, for computation styles that use denaturation, the positive design problem often requires all of the strands in the library to have similar melting temperatures, or melting temperatures above some threshold. In short, positive design tries to maximize hybridization between perfect complements.

The negative design problem requires that: (1) no strand has undesired secondary structure such as hairpin loops, (2) no string in the library hybridizes with any other string in the library, and (3) no string in the library hybridizes with the complement of any other string in the library. Thus, negative design attempts to "optimize specificity for the target structure" (144). Unplanned hybridizations can cause two types of potential errors: false positives and false negatives. False negatives occur when all (except an undetectable amount) of DNA that encodes a solution is hybridized in unproductive mismatches. Since mismatched strands are generally less stable than perfectly matched strands, false negatives can be controlled by adjusting strand concentrations. Deaton experimentally verified the occurrence of false positives, which happen when a mismatched hybridization causes a strand to be incorrectly identified as a solution (68). False positives can be prevented by ensuring that all unplanned hybridizations are unstable. In short, the negative design problem tries to minimize non-specific hybridization.

Positive design often uses GC-content and energy minimization as heuristics (see below). Negative design uses combinatorial methods (such as Hamming distance, reverse complement Hamming distance, shifted Hamming distance, and sequence symmetry minimization), and thermodynamic methods (such as minimum free energy). Constraints which incorporate both positive and negative design are probability, average incorrect nucleotides, energy gap, probability gap, and energy minimization in combination with sequence symmetry minimization. The best-performing models for designing single-strand secondary structure use simultaneous positive and negative design, and significantly outperform either method alone; however, kinetic constraints must be considered separately since low free energy does not necessarily imply fast folding (144). We believe that this same principle holds for designing hybridizations between multiple strands.

Secondary Structure of Single Strands

Most DNA computation styles need strands with no secondary structure (i.e., no tendency to hybridize with itself). There are, on the other hand, cases where specific secondary structures are desired, such as for deoxyribozyme logic gates (112); Figure 8.14 shows the desired structure. Even there, structures different from the desired must be eliminated.

There are several heuristics that are used to prevent secondary structure. Sometimes, repeated substrings and complementary substrings within a single strand which are non-overlapping and longer than some minimum length are forbidden in order to prevent stem formation. This heuristic is often called *sequence symmetry minimization* (144, 145) or *substring uniqueness* (146). Another heuristic is to forbid particular substrings; these *forbidden substrings* are usually strings known to have undesired secondary structure. For example, sequences containing GGGGG should be avoided because they may form the four-stranded G4-DNA structure (147, 148). (For more information about alternative base pairing structures see (97).) Alternatively, strands are designed using only a *three-letter alphabet* (A, C, T for DNA and A, C, U for RNA) to eliminate the potential for GC pairs which could cause unwanted secondary structure (149).

In order to design a strand with a desired secondary structure (inverse secondary structure prediction), the nucleotides at positions which bond together must be complementary. This simple approach can be improved by also requiring the strands to satisfy some free-energy-based criteria, such as those described below from Dirks et al. (144).

The *minimum free energy* constraint, which can be calculated in $O(N^3)$ time for structures with no pseudoknots (150), is used to choose sequences such that the target structure has the minimum free energy. However, since this method is negative design, it does not ensure the absence of other structures that the sequence is likely to form. Algorithms also exist to determine whether a set of strands are structure-free, where a set of sequences is considered to be structure-free if the minimum free energy of every strand in the set is greater than or equal to zero (151–153). It has also been suggested that sequences could be chosen so that the difference between the free energy of the desired structure and undesired structures is maximal (67).

The *energy minimization* constraint is used to choose sequences which have a low free energy in the target structure, but not necessarily the minimum free energy. To design strands with this constraint, first generate a random string s that satisfies the complementary requirements of the desired secondary structure. For each step (Dirks used 10^6 steps), choose a random one-point mutation. Let s' be the sequence with this random one-point mutation (and a mutation in the corresponding base required by the structure constraint, if any). Accept the mutation by replacing s with s' if:

$$e^{-\frac{\Delta G(s')-\Delta G(s)}{RT}} \geq \rho$$

where $\rho \in [0,1]$ is a random number drawn from a uniform distribution, $\Delta G(s)$ is the free energy of the sequence in secondary structure s, and $\Delta G(s')$ is the free energy of the sequence in secondary structure s' (the free energy of a given structure can be calculated in $O(N)$ time). Thus, this equation always accepts any mutations which result in no change or a decrease in free energy, and accepts with some probability any mutations which increase the free energy.

Sequences can also be chosen which maximize the *probability* of sampling the target structure. The probability $p(s)$ that every nucleotide in the sequence exactly matches the target structure s at thermodynamic equilibrium is calculated by:

$$p(s) = \frac{1}{Q} e^{-\frac{\Delta G(s)}{RT}}$$

where $\Delta G(s)$ is the free energy of the sequence in secondary structure s. The partition function, Q, is:

$$Q = \sum_{s \in \Omega} e^{-\frac{\Delta G(s)}{RT}}$$

where Ω is the set of all secondary structures that the sequence can form in equilibrium. If s^* is the target secondary structure and $p(s^*) \approx 1$, then the sequence has a high affinity and high specificity for s^*. An optimal dynamic programming algorithm calculates $p(s^*)$ for structures with no pseudoknots in $O(N^3)$ time (154), whereas $p(s^*)$ for secondary structures with pseudoknots can be calculated in $O(N^5)$ time (155).

Additionally, sequences can be chosen to minimize the *average number of incorrect nucleotides*, $n(s)$, over all equilibrium secondary structures Ω. The structure matrix S_s for a given sequence of length N in structure s is:

$$S_s[i,j] = \begin{cases} 1, \text{ if base } i \text{ is paired with base } j \text{ in } s \\ 0, \text{otherwise} \end{cases}$$

$$S_s[i,N+1] = \begin{cases} 1, \text{ if base } i \text{ is unpaired in } s \\ 0, \text{ otherwise} \end{cases}$$

where $1 \leq i \leq N$ and $1 \leq j \leq N$. The probability matrix P_s is:

$$P_s[i,j] = \sum_{s \in \Omega} p(s) S_s[i,j]$$

where $1 \leq i \leq N$ and $1 \leq j \leq N+1$. When $1 \leq j \leq N$, $P_s[i,j]$ is the probability of forming a base pair between the nucleotides at position i and j (i.e., the sum of the probabilities of each structure where i and j are paired). $P_s[i,N+1]$ is the probability that base i is unpaired. Let $n(s)$ be the average number of incorrect nucleotides over the equilibrium ensemble of secondary structures Ω. If s^* is the target structure then:

$$n(s^*) \quad = \quad N - \sum_{i=1}^{N} \sum_{j=1}^{N+1} P_s[i,j] S_{s^*}[i,j]$$

where $n(s^*)$ can be calculated in $O(N^3)$ time in structures with no pseudoknots and $O(N^5)$ in structures with pseudoknots.

Dirks et al. determined that the best-performing models are probability, average incorrect nucleotides, and energy minimization in combination with sequence symmetry minimization for the substrings that are not constrained by the desired secondary structure. The models with medium performance are the negative design methods (minimum free energy, and sequence symmetry minimization alone). The worst performing model is energy minimization (a positive design method). Surprisingly, minimum free energy performs similarly to sequence symmetry minimization; these results show that free energy measurements do not guarantee good design. An effective search must use both positive and negative design methods.

Secondary Structure of Multiple Strands

The way that DNA folds in nature is not necessarily how computers should fold DNA strands to obtain the structure, since nature has the advantage of parallel processing and the proximity of the molecules in space. The strength of a perfectly matched duplex, a positive constraint, is often estimated by either: (1) the type of hydrogen bonds, AT vs. GC, expressed as the percentage of nucleotides that are G and C bases in a strand or duplex, which is known as *GC-content*; or (2) the amount of free energy released from the formation of the hydrogen bonds and the phosphodiester bonds that hold together adjacent nucleotides in a strand. The latter model is known as the nearest-neighbor model.

Since GC base pairs are held together by three hydrogen bonds while AT base pairs are held together by only two hydrogen bonds, double-stranded DNA with a high GC content is *often* more stable than DNA with a high AT content. Many DNA library searches require each strand to have a 50% GC-content to make the thermodynamic stability of perfectly matches duplexes similar. The GC-content heuristic is simple to calculate; only the length and the number of GC bases are needed, where the length refers to the number of nucleotide base pairs. However the nearest-neighbor heuristic is more accurate than the GC-content heuristic because the nearest neighbor base stacking energies account for more of the change in free energy than the energy of the hydrogen bonding between nucleotide bases.

Requiring all pairs of strings in the library to have at least a given minimum *Hamming distance* (i.e., the number of characters in corresponding places which differ between two strings), is intended to satisfy the negative requirement that no pair of strings in the library should hybridize. A variation of this idea is the *reverse complement Hamming distance*, which is the number of corresponding positions which differ in the complement of s_1 and the reverse of s_2. This constraint is used to reduce the false positives that occur from hybridization between a word and the reverse of another word in the library.

The advantage of Hamming distance (and its variations) is its theoretical simplicity and the vast body of extant work in coding theory. Many bounds have been calculated on the optimal size of codes with various Hamming-distance-based constraints (156). Many early DNA library search algorithms used Hamming distance as a constraint to develop combinatorial algorithms based on the results from coding theory. However, Hamming distance alone is an insufficient constraint.

One problem with Hamming-distance-based heuristics is that this measure assumes that position i of the first string is aligned with position i of the second string. However, since duplexes can be formed with dangling ends and loops, this is not the only possible alignment. Various *Hamming distance slides*, *substring uniqueness* (146), partial words (157), and H-measure (71, 158) constraints have been developed to fix the alignment problem. Similarly, many of the previously mentioned constraints (such as GC-content and Hamming distance) have also been applied to windows and pairs of windows, which are substrings of a given length. Another problem with heuristics based on Hamming distance is that the percentage of matching base pairs necessary to form a duplex is not necessarily known. Melting temperature can be used to approximate what the minimum Hamming distance should be; however, for a given temperature and word set, there can be significant variation in the required minimum distance.

Now that accurate free-energy information is available for all but the most complicated secondary structures (e.g., branching loops), the nearest-neighbor model is a much more accurate method to use than the constraints based on Hamming distance. It has also been experimentally determined for a sequence A of length n and a sequence B of length m that minimum free energy is a superior constraint to BP, where

$$BP = min(n,m) - min_{-m<k<n} H(A, \sigma^k(\overline{B}))$$

where $H(*,*)$ is the Hamming distance, $\overline{B}$ is the reverse complement of B, and σ^k is the shift rightward when $k > 0$ or leftward when $k < 0$ (147) (BP is equivalent to the H-measure constraint if $n = m$). One way of using free-energy-based calculations as a constraint to prevent mismatched

duplexes is to maximize the gap between the free energy of the weakest specific hybridization and the free energy of strongest nonspecific hybridization, which we refer to as the *energy gap*; this approach was used by Penchovsky (159). A metric also exists which calculates the maximum number of stacked base pairs in any secondary structure; a thermodynamic weighting of this metric gives an upper bound on the free energy of duplex formation (160). The probability, $p(s^*)$, measurement could also be applied to duplexes. A reasonable heuristic would be to maximize the gap between the lowest probability of the desired specific hybridizations and the highest probability of undesired non-specific hybridizations, which we refer to as the *probability gap*. Algorithms exist which calculate the probability, $p(s^*)$, for all possible combinations of single and double stranded foldings between a pair of strands (161). Various equilibrium thermodynamic approaches have been used (162–166). Computational incoherence, ξ, predicts the probability of an error hybridization per-hybridization event based on statistical thermodynamics (158, 162, 167).

The physically-based models can be divided into categories based on the level of chemical detail (168). Techniques which model single molecules include molecular mechanics models such as Monte Carlo minimum free energy simulations and molecular dynamics which models the change of the system with time. Techniques which average system behavior, or mass action approaches, are less accurate but more computationally feasible. Molecular mechanics (which models the movement of the system to the lowest energy), chemical kinetics, melting temperature, and statistical thermodynamics are all mass action approaches.

Thermodynamics are best at predicting DNA structure. However, calculating these measures can be costly. According to the requirements mentioned for the negative design problem, checking that a library of size M meets specifications requires $O(M^2)$ string comparisons, where each comparison of a pair of strings of length N is potentially polynomial in N. Thus, the weaker combinatorial and heuristic predictors could be used to quickly filter a candidate set of library molecules, and then the free energy model could be used to more accurately check this set. If this approach is adopted, the correlation between these alternative heuristics and free energy measurements should be explored. Alternatively, free energy or probability approximation algorithms could be used. This approach has the advantage that techniques from randomized algorithm analysis could be used to prove the correctness of the approximation.

Melting Temperature

Melting temperature is typically used as a constraint in DNA paradigms that use multiple hybridization and denaturation steps to identify the answer, for an example see (64). When DNA is heated, the hydrogen bonds that bind two bases together tend to break apart, and the strands tend to separate from each other. The probability that a bond will break increases with temperature. This probability can be described by the melting temperature, which is the temperature in equilibrium at which 50% of the oligonucleotides are hybridized and 50% of the oligonucleotides are separated. Since temperature control is often used to help denature the strands in intermediate steps, it is advantageous for these paradigms to require all of the strands in the library to have similar melting temperatures, or melting temperatures above some threshold.

The melting temperature of a perfectly matched duplex can be roughly estimated from the 2–4 rule (67), which predicts the melting temperature as twice the number of AT base pairs plus 4 times

the number of GC base pairs. Another rough estimate of the change in melting temperature due to mismatched duplexes can also be obtained by decreasing the melting temperature of a corresponding matched duplex by 1°C per 1% mismatch; unfortunately, the inaccuracy is typically greater than 10°C (169). Neither method is recommended. A better method is to use the nearest-neighbor model regardless of whether the duplex is perfectly matched or mismatched. This method produces more accurate results because melting temperature is closely related to free energy. Melting temperature has been used to characterize the hybridization potential of a duplex (170, 171), but this measure cannot be used to predict whether two strands are bound at a given temperature since the melting temperatures of different duplexes do not necessarily correspond to relative rankings of stability.

Reaction Rates

Once the structure of candidate strands is known, the next logical question to ask is how fast do these reactions occur and what concentration is needed. Kinetics deals with the rate of change of reactions. For some implementations of DNA computers, the rate of the reaction could be an additional search constraint. System-level simulation software has been described for this purpose (172).

DNA Prediction Software

There exist many software packages that predict DNA/RNA structure, thermodynamics, or kinetics. A few well-know structure prediction software packages are: Dynalign (173), mfold (174), NUPACK (155, 175), RNAsoft (176), RNAstructure (177), and the Vienna Package (178). RNA free energy nearest neighbor parameters are available from the Turner Group (177). Some software packages which calculate thermodynamics are: HyTher (179–181), BIND (170), MELTING (182), MELTSIM (183), and MeltWin (184). Kinfold (185) simulates kinetics. EdnaCo (158) and Visual OMP (Oligonucleotide Modeling Platform; DNA Software Inc.) (186) simulate biochemical protocols *in silico*. In addition, there are many library design software packages such as: DNA Design Toolbox (187), DNASequenceCompiler (146), DNASequenceGenerator (146), NACST/Seq (188), NucleicPark (166), PERMUTE (64), PUNCH (189), SCAN (171), SEQUIN (145), SynDCode (160, 190, 191), and TileSoft (192).

8.3.2 DNA Code Design Problem

Once the desired constraints are known, how should one design a sequence generator to find strings that satisfy those constraints? A good generator should be reliable, extensible, efficient, and scalable. Ideally the generator should find as large a set as possible, work for multiple problems, and should allow constraints to be added and removed easily. However comparisons of sequence generation algorithms are difficult because the algorithms are usually written and tested for a specific DNA computation problem and specific set of constraints; an algorithm that does well on one constraint set may not do well on another constraint set. Thus in this section we briefly explain

several approaches to give the flavor of possible solutions to the DNA Code Design Problem; see also (143).

Early algorithms to find DNA word sets focused on the Hamming distance constraint or variations thereof to achieve a theoretical abstraction of the constraints, which allowed the use of combinatorial algorithms (e.g., (69)) and proofs of completeness (i.e., that the size of the pool is optimal or near optimal) (156). However, in the process the constraints are simplified so much that they no longer accurately predict DNA structure. Current algorithms tend to use a more complex combination of the constraints. However, since these constraints are difficult to abstract, more recent programs resort to genetic algorithms, random search, exhaustive search, and local stochastic search algorithms.

Combinatoric Algorithms

Because of the association between DNA code design and coding theory, early algorithms tended to focus on finding optimal code sizes. Many proofs have been found which bound the size of optimal codes for simple combinations of constraints based on Hamming distance and reverse complement Hamming distance (156). These proofs can be used to evaluate the optimality of a solution to the DNA Code Design Problem. Algebraic properties, formal language theory, and coding theory have also been used to show properties of DNA-compliant languages (193). However, the tradeoff is that many of these proofs are extremely difficult to extend to complex combinations of constraints that model the physical world more realistically. As a result, these algorithms tended to be deterministic, combinatorial, and specific to the DNA computer that they were designed for.

For example, the "template-map" strategy (69) obtains a large number of dissimilar word sequences from a significantly smaller number of of templates and maps *using theoritical proofs*, where a template is a string chosen from the alphabet $\{A,C\}$ and a map is a string of the same length chosen from the alphabet $\{0,1\}$. When a map m is applied to a template t, a character in the template, t_i, is replaced with its complement if the corresponding character in the map, m_i, is 1; if m_i is 0 then there is no change to t_i (e.g., when the map 10100101 is applied on template AACCACCA, it produces the string TAGCAGCT.) Since each template and map pair uniquely describes a string from the alphabet $\{A,C,G,T\}$, additional constraints are needed to prevent non-specific hybridizations. The templates are also required to be "conflict-free", where two templates are considered to be conflict-free if they generate two strings which have a Hamming distance and reverse complement Hamming distance of at least 4 when paired with any two maps. In addition, the template and map pairs are also required to generate strings with a 50% GC-content. The obvious limitation of this method is with respect to extensibility and scalability.

Randomized Algorithms

Later algorithms tend to focus on being extensible to a variety of problems and constraints and also on accurately modeling the physical world; this trend can be seen in the current discussions about defining a standard for biomolecular computing simulation software (194). Since the search space is large and the constraints are complex, most of the randomized algorithms used for DNA code design tend to be Las Vegas algorithms (algorithms which vary in run time) and not Monte

Carlo algorithms (algorithms which sometimes produce incorrect answers); thus the efficiency with which a randomized algorithm finds or converges to a solution is an important consideration for evaluating these types of algorithms. In addition, these algorithms may also vary in solution quality from run to run, so the quality of the solution is also important.

The PERMUTE program (64) is an example of a simple randomized algorithm. It generates random nucleotides from the three letter alphabet $\{A,C,U\}$ and then permutes the sequence until the constraints are satisfied. If no permutation produces a valid string, then a new random string is generated. A simple variation on this idea is to generate a random candidate string, add the string to the pool only if it satisfies the constraints, and repeat (195). These types of "generate-and-test" algorithms perform well in situations where the search process does not tend to get stuck in local minima. However the constraints must be set appropriately before algorithm executes and the generator can not suggest whether it is possible to find better sets which satisfy the same constraints.

The DNASequenceGenerator (146) is an example of a slightly more complicated randomized algorithm. This algorithm generates a pool of n_b-unique sequences from a directed graph whose nodes are labeled with sequences of length n_b, which are referred to as "base strands". A directed edge, $\overrightarrow{(u,v)}$, connects node u and v if the last n_b-1 characters of base strand u are the same as the first n_b-1 characters of base strand v. Thus a string of length n_s is represented by a path of length (n_s-n_b+1); the set of paths of length (n_s-n_b+1) which do not share any nodes corresponds to a set of n_b-unique sequences of length n_s. The nodes of certain base strands (such as self-complementary substrings, forbidden substrings, substrings containing two consecutive GG or CC bases, substrings containing specified GC-content, etc.) can be restricted by marking their corresponding nodes as forbidden or by removing them from the graph. In each iteration, the algorithm randomly chooses a start node and performs a random walk to find a path of length (n_s-n_b+1) which does not contain forbidden nodes, nodes used in other paths, or the reverse complement of nodes used in other paths. If a complete path which satisfies the constraints (such as melting temperature and GC-content) is found, the sequence is added to the library of strings, otherwise the walk backtracks and attempts to find another path. A limitation of this algorithm is that a large amount of memory may be needed to store the graph.

Most current research in DNA word design falls in the category of stochastic local search algorithms (which includes the evolutionary algorithms described below). Stochastic local search algorithms (SLS) are the subset of randomized algorithms which make use of the previous randomized choices when generating or selecting new candidate solutions. More specifically, "the local search process is started by selecting an initial candidate solution, and then proceeds by iteratively moving from one candidate solution to a neighboring candidate solution, where the decision on each search step is based on a limited amount of local information only. In stochastic local search algorithms, these decisions as well as the initial search initialization can be randomized" (196). Many SLS algorithms have parameters which need to be set manually. The comparison of these algorithms can be misleading when the parameter settings are unevenly optimized; thus care must be taken to ensure that the parameters are equally optimized or that at least the same amount of effort is spent on each algorithm to optimize the parameters if the optimal settings are uncertain.

Given a set of individual and pairwise constraints on strands (e.g., Hamming distance, reverse

Hamming distance, GC-content, or thermodynamics), the SLS-THC algorithm (196–199) begins with a randomly chosen pool of strings of size N, where each string is of length n and each string satisfies any constraints specified on individual strings. To obtain good performance, the algorithm stores the results of the calculations for the pairwise constraints in a table; thus modifying a word in the pool requires only $\Theta(N)$ calculations. In each iteration, the algorithm picks a pair of words (uniformly at random) that has a conflict (a violation of a pairwise search constraint) and modifies one of the words. All single-base mutations to each string in the conflicting pair which satisfy the individual constraints (the 1-mutation neighborhood) are considered as modifications. With constant probability θ, a modification in the 1-mutation neighborhood is chosen at random, otherwise a modification is chosen which maximally reduces the number of pair conflicts in the pool. Empirical analysis of the run-time distributions of the algorithm on hard design problems indicates that the search performance is compromised by stagnation; this problem can be overcome by the occasional random replacement of a small fraction of the strings in the pool (197). The algorithm terminates if S has no conflicts or if a specified number of iterations have been completed. If the algorithm terminates before it finds a valid set of size N then a word in a conflicting pair is randomly deleted from the pool until no conflicts remain.

The SLS-THC algorithm is a more sophisticated search than the previous randomized algorithms because it utilizes local information in its search process. The search process can be thought of as a conflict-directed random walk. As the algorithm runs, at any given time there may be pairwise conflicts in the pool; allowing these conflicts to remain may help the algorithm overcome local minima because the decision of which conflicting string to remove is delayed. Since every conflicting pair has the same probability of being mutated in each iteration, there is high probably that strings that create minor conflicts will be resolved by only a few mutations and a high probability that strings which prevent the pool size from growing (local minima) will be mutated greatly or even replaced. It has been empirically demonstrated that the SLS-THC algorithm matches or improves upon the pool sizes obtained from the best known theoretical constructions for several different combinations of Hamming distance, reverse Hamming distance, and GC-content constraints (196).

Evolutionary algorithms (EA) are a subset of SLS algorithms which use techniques inspired by biological evolution. The solution pool is represented by a population of 'individuals' or 'chromosomes'. EAs use selection, mutation, and recombination on the population to utilize local information and prevent local minima in order to efficiently optimize the population. There are several types of evolutionary algorithms such as genetic algorithms (GA) (200), evolution strategies (201, 202), and evolutionary programming (203). However since current work often blends concepts from many styles of EAs, we do not emphasize the differences between the types of EAs.

The goal of a GA is to minimize or maximize a measure of fitness; this concept corresponds to the biological concept of "selection of the fittest". For example, in some GA implementations of the DNA word design problem, the fitness is based on the Hamming distance between strings (68, 204) or based on the partition function (205). Other GAs have used a single fitness function which incorporates multiple constraints (195); as a result, several experimental runs may be required to decide how to set the parameters. When the constraints are mutually independent, the parameter values can be determined independently. However, in the DNA word design problem, it is often the

case that optimizing one constraint causes a relative tradeoff in the optimality of another constraint (e.g., the chance of non-specific hybridizations can be reduced by using only the three bases A, T, and C, but this technique also increases the similarity of the strings). When the parameters are not mutually exclusive, finding the optimal parameters settings can be difficult (196). It has been suggested that as the number of design constraints is increased, a single fitness measure which incorporates all of the design constraints may not be appropriate for the DNA word design problem because the relative importance of each constraint is often unknown (195). Some more recent GAs, such as NACST/Seq (188,206,207), attempt to resolve these problems using a multi-objective GA.

8.3.3 DNA Code Evaluation Problem

Of the heuristics previously mentioned, the most appropriate method for obtaining an estimate of the absolute or relative rate of hybridization error is thermodynamics and statistical thermodynamics. For example, $p(s^*)$, $n(s^*)$, pair probabilities, and free energy have been used to evaluate whether a singly stranded sequence will form a desired secondary structure, s^* (144). Statistical thermodynamics (the partition function of all hybridized configurations) have been used to predict the error rate in the set of strands used in Adleman's original Hamiltonian Path problem (205). Computational incoherence (162, 167), ξ, could also be used for evaluation. In addition, the energy gap or probability gap could be used for evaluation (199). The most significant evaluation criterion is how the strands perform in the laboratory, since this is what the library is ultimately designed for.

Research in DNA libraries has two main goals: (1) to further understand DNA chemistry, and (2) to understand search techniques useful for constructing sets of DNA codes. Although there is a growing consensus that DNA computers will never be as practical or as fast as conventional computers, biological computers have the advantage that their style of computation is closer to natural processes. Deaton states that the process of converting an algorithm into a biomolecular systems "is as difficult [i.e., NP-hard or harder] as the combinatorial optimization problems they are intended to solve" (142). However, successful research in DNA libraries will help to reduce errors in DNA computation and may discover new information about how DNA interacts with itself. Although current DNA computers are simplistic in comparison to natural biochemical processes, DNA computation may help to develop alternative theories for how cells work or could have evolved (208). In addition, research in DNA design also pertains to DNA nanotechnology, PCR-based applications, and DNA arrays. Breakthroughs in this field will add to the current knowledge of DNA chemistry as well as DNA computers.

8.3.4 Exploiting Inexact Matching

In the preceding, we assumed that the applications to which the designed word sets will be put require exact matching for correctness of operation. This is indeed true of combinatorial DNA computing, to avoid false positives, i.e., spurious solutions, and it is somewhat true in enzymatic DNA computing, to minimize cross-talk between signals. On the other hand, there can be an array of applications that inherently allow modest amounts of error. Such is the case with signal processing applications, where the input data are noisy. It is preferable in such situations to allow

imperfect matches, i.e., to build the possibility of imperfect matches directly into the design of the word set.

Tsaftaris (209, 210) considers a hypothetical scenario in which a database of signals is stored as a pool of DNA. Each signal is represented as a double-stranded section of DNA. The database allows matching queries, in which one asks if a given (short) probe signal is approximately equal to some portion of one of the stored (target) signals; the target signal and the position of the match are identified. To run the matching query, a sample of the database is denatured, the probe is represented as the complementary oligonucleotide, hybridization is allowed to take place, and then the result is isolated. In such a setting, it is advantageous explicitly to allow some degree of hybridization errors between strands that encode *adjacent* signal levels. The word design problem is then not just that of choosing some N oligonucleotides of a given length, but of assigning them to the N discrete signal levels in such a way that for signal levels that are close to one another, the likelihood of a stable mismatch is inversely proportional to the level difference, and for signal levels exceeds some threshold, that likelihood is negligible. This is called the *noise tolerance constraint*, and is imposed in addition to the usual combinatorial constraints. A stochastic algorithm that builds upon thermodynamic models of SantaLucia (180) is proposed by Tsaftaris and demonstrated for $N = 128$ and 10nt oligonucleotides (209).

8.4 Conclusion

This review focused on a few selected topics in nanocomputing. The literature grows by the day. For combinatorial approaches, which predominated at the outset of DNA computing research era, consult, e.g., the review (99). For state-machine based approaches, predicted at least as early as in the work of Manin, initiated by Rothemund, and forcefully demonstrated by Benenson (211, 212), in which finite control is achieved using collections of customized enzymes, consult, e.g., the review (213). For cell and membrane computing, consult (85, 90, 214–216). For recent achievements in self-assembly, in particular assembly of almost arbitrary planar shapes, see (217). For recent achievements in enzymatic computing, see (218). For architectural advances, spearheaded by dyed-in-the wool computer scientists, see e.g. (219, 220).

1. International Technology Roadmap for Semiconductors, 2003.

2. Bennett CH. The thermodynamics of computation - a review. Int J Theoretical Physics 1982; 21:905–940.

3. Zhirnov VV, Cavin RK III, Hutchby JA, Bourianoff GI. Limits to binary logic switching—a gedanken model. Proceedings of the IEEE 2003;91:1934–1939.

4. Ball P. Chemistry meets computing. Nature 2000;406:118–120.

5. Reed M, Tour JM. Computing with molecules. Scientific American 2000;86–93.

6. Zhirnov VV, Herr DJC. New frontiers: Self-assembly and nanoelectronics. IEEE Computer 2001;34.

7. Hutchby JA, Bourianoff GI, Zhirnov VV, Brewer JE. Extending the road beyond CMOS. IEEE Circuits and Devices Magazine 2002;0.

8. Bourianoff G. The future of nanocomputing. IEEE Computer 2003;36.

9. Stone NJ, Ahmed H. Silicon single electron memory cell. Applied Physics Letters 1998; 73:2134–2136.

10. Mahapatra S, Vish V, Wasshuber C, Banerjee K, Ionescu AM. Analytical modeling of single electron transistor for hybrid CMOS-SET analog IC design. IEEE Transactions on Electron Devices 2004;51:1772–1782.

11. Bachtold A, Hadley P, Nakanishi T, Dekker C. Logic circuits with carbon nanotube transistors. Science 2001;294:1317–1320.

12. Chen Y, Ohlberg DAA, Medeiros-Ribeiro G, Chang YA, Williams RS. Self-assembled growth of epitaxial erbium disilicide nanowires. Applied Physics Letters 2000;76:4004–4006.

13. Cui Y, Wei Q, Park H, Lieber CM. Nanowire nanosensors for highly sensitive and selective detection of biological and chemical species. Science 2001;293:1289–1292.

14. Huang Y, Duan X, Cui Y, Lauhon LJ, Kim KH, Lieber CM. Logic gates and computation from assembled nanowire building blocks. Science 2001;294:1313–1317.

15. Reed MA, Zhou C, Muller CJ, Burgin TP. Conductance of a molecular junction. Science 1997;278:252–254.

16. Reed M, Chen J, Rawlett AM, Price DW, Tour JM. Molecular random access memory cell. Applied Physics Letters 2001;78:3735–3737.

17. Chen Y, Ohlberg DAA, Li X, et al. Nanoscale molecular-switch devices fabricated by imprint lithography. Applied Physics Letters 2003;82:1610–1612.

18. Csaba G, Imre A, Bernstein GH, Porod W, Metlushko V. Nanocomputing by field-coupled nanomagnets. IEEE Transactions on Nanotechnology 2002;1:209–213.

19. Porod W, Lent CS, Bernstein GH, et al. Quantum-dot cellular automata: computing with coupled quantum dots. International Journal of Electronics 1999;86:549–590.

20. Heath JR, Kuekes PJ, Snider GS, Williams RS. A defect-tolerant computer architecture: Opportunities for nanotechnology. Science 1998;280:1716–1721.

21. Collier CP, Wong EW, Belohradský M, et al. Electronically configurable molecular-based logic gates. Science 1999;285:391–394.

22. Metzger RM. Electrical rectification by a molecule: The advent of unimolecular electronic devices. Accounts of Chemical Research 1999;32:950–957.

23. Ellenbogen JC, Love JC. Architectures for molecular electronic computers: 1. Logic structures and an adder built from molecular electronic diodes. Proceedings of the IEEE 2000; 88:386–426.

24. Joachim C, Gimzewski JK, Aviram A. Electronics using hybrid-molecular and monomolecular devices. Nature 2000;408:541–548.

25. Donhauser ZJ, Mantooth BA, Kelly KF, et al. Conductance switching in single molecules through conformational changes. Science 2001;292:2303–2307.

26. Goldstein SC, Budiu M. NanoFabrics: Spatial computing using molecular electronics. In: Proceedings of the 28th International Symposium on Computer Architecture 2001. 2001 .

27. Pease AR, Jeppesen JO, Stoddart JF, Luo Y, Collier CP, Heath JR. Switching devices based on interlocked molecules. Accounts of Chemical Research 2001;34:433–444.

28. Postma HWC, Teepen T, Yao Z, Grifoni M, Dekker C. Carbon nanotube single-electron transistors at room temperature. Science 2001;293:76–79.

29. Mishra M, Goldstein SC. Scalable defect tolerance for molecular electronics. In: 1st Workshop on Non-Silicon Computing. Cambridge, MA, 2002 .

30. de Silva AP, Gunaratne HQN, McCoy CP. A molecular photoionic AND gate based on fluorescent signalling. Nature 1993;364:42–44.

31. de Silva AP, Gunaratne HQN, McCoy CP. Molecular photoionic AND logic gates with bright fluorescence and "off-on" digital action. Journal of the American Chemical Society 1997; 119:7891–7892.

32. Credi A, Balzani V, Langford SJ, Stoddart JF. Logic operations at the molecular level. An XOR gate based on a molecular machine. Journal of the American Chemical Society 1997; 119:2679–2681.

33. Pina F, Melo MJ, Maestri M, Passaniti P, Balzani V. Artificial chemical systems capable of mimicking some elementary properties of neurons. Journal of the American Chemical Society 2000;122:4496–4498.

34. de Silva AP, McClenaghan ND. Proof-of-principle of molecular-scale arithmetic. Journal of the American Chemical Society 2000;122:3965–3966.

35. Banerjee K, Soukri SJ, Kapur P, Saraswat K. 3-D ICs: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. Proceedings of the IEEE 2001;89:602–633.

36. Bilardi G, Preparata FP. Horizons of parallel computation. Tech. Rep. CS-93-20, Department of Computer Science, Brown University, 1993.

37. Abelson H, Allen D, Coore D, et al. Amorphous computing. Communications of the ACM 2000;43:74–82.

38. Goldstein SC, Rosewater D. Digital logic using molecular electronics. In: IEEE International Solid-State Circuits Conference. San Francisco, CA, 2002 12.5.

39. Gruau F, Malbos P. The blob: A basic topological concept for hardware-free distributed computation. In: Unconventional Models of Computation, Third International Conference, UMC 2002, Kobe, Japan, October 15-19, 2002, Proceedings, Calude C, Dinneen MJ, Peper F, eds., vol. 2509 of *Lecture Notes in Computer Science*. Springer, 2002 151–163.

40. Gruau F, Lhuillier Y, Reitz P, Temam O. Blob computing. In: Computing Frontiers 2004 ACM SIGMicro. 2004 .

41. Durbeck LJK, Macias NJ. The Cell Matrix: an architecture for nanocomputing. Nanotechnology 2001;12:217–230.

42. Munakata T, Sinha S, Ditto WL. Chaos computing: Implementation of fundamental logic gates by chaotic elements. IEEE Transactions on Circuits and Systems—I: Fundamental Theory and Applications 2002;49:1629–1633.

43. Adleman LM. Molecular computation of solutions to combinatorial problems. Science 1994; 266:1021–1024.

44. Deaton RJ, Garzon M, Rose JA, Franceschetti DR, Stevens SE Jr. DNA computing: A review. Fundamenta Informaticae 1998;35:231–245.

45. Lipton RJ. DNA solution of hard computational problems. Science 1995;268:542–545.

46. Ruben AJ, Landweber LF. Timeline: The past, present and future of molecular computing. Nature Reviews Molecular Cell Biology 2000;1:69–72.

47. Wang L, Liu Q, Corn RM, Condon AE, Smith LM. Multiple word DNA computing on surfaces. Journal of the American Chemical Society 2000;122:7435–7440.

48. Winfree E. On the computational power of DNA annealing and ligation. In: Lipton and Baum (221) 199–221.

49. Winfree E. Complexity of restricted and unrestricted models of molecular computation. In: Lipton and Baum (221) 187–198.

50. Watson J, Crick FHC. A structure for deoxyribose nucleic acid. Nature 1953;171:737.

51. LaBean TH, Yan H, Kopatsch J, et al. Construction, analysis, ligation, and self-assembly of DNA triple crossover complexes. Journal of the American Chemical Society 2000;122:1848–1860.

52. Watson JD, Hopkins NH, Roberts JW, Steitz JA, Weiner AM. Molecular Biology of the Gene. 4th ed. Menlo Park, CA: Benjamin/Cummings, 1988.

53. Winfree E, Liu F, Wenzler LA, Seeman NC. Design and self-assembly of two-dimensional DNA crystals. Nature 1998;394:539–544.

54. Wang L, Hall JG, Lu M, Liu Q, Smith LM. A DNA computing readout operation based on structure-specific cleavage. Nature Biotechnology 2001;19:1053–1059.

55. Braich RS, Chelyapov N, Johnson C, Rothemund PWK, Adleman L. Solution of a 20-variable 3-SAT problem on a DNA computer. Science 2002;296:499–502.

56. Morimoto N, Arita M, Suyama A. Solid phase DNA solution to the Hamiltonian path problem. In: Rubin and Wood (222) 193–206.

57. Ouyang Q, Kaplan PD, Liu S, Libchaber A. DNA solution of the maximal clique problem. Science 1997;278:446–449.

58. Pirrung MC, Connors RV, Odenbaugh AL, Montague-Smith MP, Walcott NG, Tollett JJ. The arrayed primer extension method for DNA microchip analysis. Molecular computation of satisfaction problems. Journal of the American Chemical Society 2000;122:1873–1882.

59. Garzon M, Gao Y, Rose JA, et al. In vitro implementation of finite-state machines. In: Proceedings 2nd International Workshop on Implementing Automata WIA'97, vol. 1436 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Heidelberg, New York., 1998 56–74.

60. Guarnieri F, Fliss M, Bancroft C. Making DNA add. Science 1996;273:220–223.

61. Hug H, Schuler R. DNA-based parallel computation of simple arithmetic. In: Jonoska and Seeman (223) .

62. Mao C, LaBean TH, Reif JH, Seeman NC. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. Nature 2000;407:493–496. Erratum, *Nature 408* (2000), 750.

63. Rothemund PWK, Winfree E. The program-size complexity of self-assembled squares. In: STOC'00: The Thirty-Second Annual ACM Symposium on Theory of Computing. 2000 .

64. Faulhammer D, Cukras AR, Lipton RJ, Landweber LF. Molecular computation: RNA solutions to chess problems. Proceedings of the National Academy of Sciences of the USA (PNAS) 2000;97:1385–1389. The PERMUTE Program is available at `http://www.pnas.org/cgi/content/full/97/4/1385/DC1`.

65. Hartmanis J. On the weight of computation. Bulletin of the EATCS 1995;55:136–138.

66. Baum EB. DNA sequences useful for computation. In: Landweber and Baum (224) 235–241.

67. Brenneman A, Condon AE. Strand design for bio-molecular computation. Tech. rep., University of British Columbia, 2001.

68. Deaton RJ, Murphy RC, Garzon M, Franceschetti DR, Stevens SE Jr. Good encodings for DNA-based solutions to combinatorial problems. In: Landweber and Baum (224) 247–258.

69. Frutos AG, Liu Q, Thiel AJ, et al. Demonstration of a word design strategy for DNA computing on surfaces. Nucleic Acids Research 1997;25:4748–4757.

70. Garzon M, Deaton RJ, Niño LF, Stevens E, Wittner M. Encoding genomes for DNA computing. In: Gemetic Programming 1998: Proceedings 3rd Genetic Programming Conference. Morgan Kaufmann, 1998 684–690.

71. Garzon M, Neathery P, Deaton RJ, Murphy RC, Franceschetti DR, Stevens SE Jr. A new metric for DNA computing. In: Proceedings 2nd Genetic Programming Conference. 1997 472–478.

72. Marathe A, Condon AE, Corn RM. On combinatorial DNA word design. In: Winfree and Gifford (225) 75–89. Extended abstract.

73. Reinert G, Schbath S, Waterman MS. Probabilistic and statistical properties of words: An overview. Journal of Computational Biology 2000;7:1–46.

74. Feldkamp U, Banzhaf W, Rauhe H. A DNA sequence compiler. Tech. rep., University of Dortmund, 2000.

75. Hug H, Schuler R. Strategies for the development of a peptide computer. Bioinformatics 2001;17:364–368.

76. Sakamoto K, Gouzu H, Komiya K, et al. Molecular computation by DNA hairpin formation. Science 2000;288:1223–1226.

77. Winfree E. Simulations of computing by self-assembly. In: Kari et al. (226) 213–242.

78. Basu S, Karig D, Weiss R. Engineering signal processing in cells: Towards molecular concentration band detection. In: Hagiya and Ohuchi (227) .

79. Conrad M. On design principles for a molecular computer. Communications of the ACM 1985;28:464–480.

80. Guet CC, Elowitz MB, Wang W, Leibler S. Combinatorial synthesis of genetic networks. Science 2002;296:1466–1470.

81. Hayes B. Computing comes to life. American Scientist 2001;89:204–208.

82. Ji S. The cell as the smallest DNA-based molecular computer. BioSystems 1999;52:123–133.

83. Knight TF Jr, Sussman GJ. Cellular gate technology. In: Proceedings UMC98, First International Conference on Unconventional Models of Computation. 1998 .

84. LaBean TH, Winfree E, Reif JH. Experimental progress in computation by self-assembly of DNA tilings. In: Winfree and Gifford (225) 123–140.

85. Landweber LF, Kari L. The evolution of cellular computing: nature's solution to a computational problem. BioSystems 1999;52:3–13.

86. Landweber LF, Kuo TC, Curtis EA. Evolution and assembly of an extremely scrambled gene. Proceedings of the National Academy of Sciences of the USA 2000;97:3298–3303.

87. Reif JH. Parallel biomolecular computation. In: Rubin and Wood (222) 217–254.

88. Saylor G. Construction of genetic logic gates for biocomputing. In: 101st General Meeting of the American Society for Microbiology. 2001 .

89. Weiss R. Cellular Computation and Communication using Engineered Genetic Regulatory Networks. Ph.D. thesis, Massachusetts Institute of Technology, 2001.

90. Weiss R, Basu S. The device physics of cellular logic gates. In: First Workshop on Non-Silicon Computing. 2002 .

91. Weiss R, Homsy G, Nagpal R. Programming biological cells. Tech. rep., MIT Laboratory for Computer Science and Artificial Intelligence, 1998.

92. Weiss R, Homsy GE, Knight TF Jr. Towards *in vivo* digital circuits. In: DIMACS Workshop on Evolution as Computation. 1999 .

93. Winfree E, Yang X, Seeman NC. Universal computation via self-assembly of DNA: Some theory and experiments. In: Landweber and Baum (224) 191–213. Errata: http://www.dna.caltech.edu/Papers/self-assem.errata.

94. Cox JC, Ellington AD. DNA computation function. Current Biology 2001;11:R336.

95. Yurke B, Mills Jr AP, Cheng SL. DNA implementation of addition in which the input strands are separate from the operator strands. BioSystems 1999;52:165–174.

96. Reif JH. DNA lattices: A method for molecular scale patterning and computation. Computer and Scientific Engineering Magazine 2002;4:32–41.

97. Seeman NC. It started with Watson and Crick, but it sure didn't end there: Pitfalls and possibilities beyond the classic double helix. Natural Computing: an international journal 2002;1:53–84.

98. Wang H. Proving theorems by pattern recognition I. Commun ACM 1960;3:220–234.

99. Jonoska N, Kephard DE, Lefevre J. Trends in computing with DNA. J Comput Sci Technol 2004;19:98.

100. Carbone A, Mao C, Constantinou PE, et al. 3D fractal DNA assembly from coding, geometry and protection. Natural Computing 2004;3:235–252.

101. Barish RD, Rothemund PWK, Winfree E. Two computational primitives for algorithmic self-assembly: Copying and counting. Nano Letters 2005;5:2586–2592.

102. Winfree E. DNA computing by self-assembly. National Academy of Engineering's The Bridge 2003;33:31–38.

103. Schulman R, Winfree E. Programmable control of nucleation for algorithmic self-assembly. In: Ferretti et al. (228) 319–328.

104. Chen HL, Goel A. Error free self-assembly using error prone tiles. In: Ferretti et al. (228) 62–75.

105. Winfree E, Bekbolatov R. Proofreading tile sets: Error-correction for algorithmic self-assembly. In: Chen and Reif (229) 126–144.

106. Reif JH, Sahu S, Yin P. Compact error-resilient computational DNA tiling assemblies. In: Ferretti et al. (228) 293–307.

107. Stojanovic MN, de Prada P, Landry DW. Catalytic molecular beacons. ChemBioChem 2001; 2:411–415.

108. Stojanovic MN, Mitchell TE, Stefanovic D. Deoxyribozyme-based logic gates. Journal of the American Chemical Society 2002;124:3555–3561.

109. Stojanovic MN, Kolpashchikov D. Modular aptameric sensors. Journal of the American Chemical Society 2004;126:9266–9270.

110. Stojanovic MN, Semova S, Kolpashchikov D, Morgan C, Stefanovic D. Deoxyribozyme-based ligase logic gates and their initial circuits. Journal of the American Chemical Society 2005;127:6914–6915.

111. Stojanovic MN, Stefanovic D. Deoxyribozyme-based half adder. Journal of the American Chemical Society 2003;125:6673–6676.

112. Stojanovic MN, Stefanovic D. A deoxyribozyme-based molecular automaton. Nature Biotechnology 2003;21:1069–1074.

113. Andrews B. Games, Strategies, and Boolean Formula Manipulation. Master's thesis, University of New Mexico, 2005.

114. Epstein IR, Pojman JA. An Introduction to Nonlinear Chemical Dynamics. New York: Oxford University Press, 1998.

115. Field RJ, Körös E, Noyes R. Oscillations in chemical systems. II. Thorough analysis of temporal oscillation in the bromate-cerium-malonic acid system. Journal of the American Chemical Society 1972;94:8649–8664.

116. Noyes R, Field RJ, Körös E. Oscillations in chemical systems. I. Detailed mechanism in a system showing temporal oscillations. Journal of the American Chemical Society 1972; 94:1394–1395.

117. Tyson JJ. The Belousov-Zhabotinskii Reaction, vol. 10 of *Lecture Notes in Biomathematics*. Berlin: Springer-Verlag, 1976.

118. Hjelmfelt A, Ross J. Chemical implementation and thermodynamics of collective neural networks. Proceedings of the National Academy of Sciences of the USA 1992;89:388–391.

119. Hjelmfelt A, Ross J. Pattern recognition, chaos, and multiplicity in neural networks of excitable systems. Proceedings of the National Academy of Sciences of the USA 1994;91:63–67.

120. Hjelmfelt A, Schneider FW, Ross J. Pattern recognition in coupled chemical kinetic systems. Science 1993;260:335–337.

121. Hjelmfelt A, Weinberger ED, Ross J. Chemical implementation of neural networks and Turing machines. Proceedings of the National Academy of Sciences of the USA 1991;88:10983–10987.

122. Hjelmfelt A, Weinberger ED, Ross J. Chemical implementation of finite-state machines. Proceedings of the National Academy of Sciences of the USA 1992;89:383–387.

123. Laplante JP, Pemberton M, Hjelmfelt A, Ross J. Experiments on pattern recognition by chemical kinetics. The Journal of Physical Chemistry 1995;99:10063–10065.

124. Rössler OE. A principle for chemical multivibration. Journal of Theoretical Biology 1972; 36:413–417.

125. Rössler OE, Seelig FF. A Rashevsky-Turing system as a two-cellular flip-flop. Zeitschrift für Naturforschung 1972;27 b:1444–1448.

126. Seelig FF, Rössler OE. Model of a chemical reaction flip-flop with one unique switching input. Zeitschrift für Naturforschung 1972;27 b:1441–1444.

127. Szilard L. Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen. Zeitschrift für Physik 1929;53:840–856.

128. Matías MA, Güémez J. On the effects of molecular fluctuations on models of chemical chaos. Journal of Chemical Physics 1995;102:1597–1606.

129. Moore C. Unpredictability and undecidability in dynamical systems. Physical Review Letters 1990;64:2354–2357.

130. Wolfram S. Undecidability and intractability in theoretical physics. Physical Review Letters 1985;54:735–738.

131. Winfree AT. Spiral waves of chemical activity. Science 1972;175:634–635.

132. Steinbock O, Kettunen P, Showalter K. Anisotropy and spiral organizing centers in patterned excitable media. Science 1995;269:1857–1860.

133. Steinbock O, Kettunen P, Showalter K. Chemical wave logic gates. Journal of Physical Chemistry 1996;100:18970–18975.

134. Steinbock O, Toth A, Showalter K. Navigating complex labyrinths: Optimal paths from chemical waves. Science 1995;267:868–871.

135. Yurke B, Turberfield AJ, Mills AP Jr, Neumann JL. A molecular machine made of and powered by DNA. In: The 2000 March Meeting of the American Physical Society. 2000 .

136. Magnasco MO. Molecular combustion motors. Physical Review Letters 1994;72:2656–2659.

137. Magnasco MO. Chemical kinetics is Turing universal. Physical Review Letters 1997; 78:1190–1193.

138. Homsy GE. Performance limits on biochemical computation. Tech. rep., MIT Artificial Intelligence Laboratory, 2000.

139. Hiratsuka M, Aoki T, Higuchi T. Enzyme transistor circuits for reaction-diffusion computing. IEEE Transactions on Circuits and systems—I: Fundamental Theory and Applications 1999; 46:294–303.

140. Morgan C, Stefanovic D, Moore C, Stojanovic MN. Building the components for a biomolecular computer. In: Ferretti et al. (230) .

141. Farfel J, Stefanovic D. Towards practical biomolecular computers using microfluidic deoxyribozyme logic gate networks. In: Carbone et al. (231) 221–232.

142. Deaton RJ, Garzon M. Thermodynamic constraints on DNA-based computing. In: Păun (216) 138–152.

143. Mauri G, Ferretti C. Word design for molecular computing: A survey. In: Chen and Reif (229) 37–47.

144. Dirks RM, Lin M, Winfree E, Pierce NA. Paradigms for computational nucleic acid design. Nucleic Acids Research 2004;32:1392–1403.

145. Seeman NC. *De Novo* design of sequences for nucleic acid structural engineering. Journal of Biomolecular Structure & Dynamics 1990;8:573–581.

146. Feldkamp U, Rauhe H, Banzhaf W. Software tools for DNA sequence design. Genetic Programming and Evolvable Machines 2003;4:153–171.

147. Tanaka F, Kameda A, Yamamoto M, Ohuchi A. Specificity of hybridization between DNA sequences based on free energy. In: Carbone et al. (231) 366–375.

148. Sen D, Gilbert W. Formation of parallel four-stranded complexes by guanine-rich motifs in DNA and its implications for meiosis. Nature 1988;334:364–366.

149. Mir KU. A restricted genetic alphabet for DNA computing. In: Landweber and Baum (224) .

150. Zuker M, Stiegler P. Optimal computer folding of large RNA sequences using thermodynamics and auxiliary information. Nucleic Acids Research 1981;9:133–148.

151. Andronescu M, Dees D, Slaybaugh L, et al. Algorithms for testing that sets of DNA word designs avoid unwanted secondary structure. In: Hagiya and Ohuchi (227) 182–195.

152. Kobayashi S. Testing structure freeness of regular sets of biomolecular sequences (extended abstract). In: Ferretti et al. (228) 192–201.

153. Kijima A, Kobayashi S. Efficient algorithm for testing structure freeness of finite set of biomolecular sequences. In: Carbone et al. (231) 278–288.

154. McCaskill JS. The equilibrium partition function and base pair binding probabilities for RNA secondary structure. Biopolymers 1990;29:1105–1119.

155. Dirks RM, Pierce NA. A partition function algorithm for nucleic acid secondary structure including pseudoknots. Journal of Computational Chemistry 2003;24:1664–1677. NUPACK is available at `http://www.acm.caltech.edu/~niles/software.html`.

156. Marathe A, Condon AE, Corn RM. On combinatorial DNA word design. Journal of Computational Biology 2001;8:201–220.

157. Leupold P. Partial words for DNA coding. In: Ferretti et al. (230) .

158. Garzon M, Deaton RJ, Rose JA, Lu L, Franceschetti DR. Soft molecular computing. In: *Proc. DNA5-99 Workshop, AMS DIMACS Series in Theoretical Computer Science* (225), 91–100. EdnaCo is available at `http://zorro.cs.memphis.edu/~cswebadm/csweb/research/pages/bmc/` or `http://engronline.ee.memphis.edu/molec/demos.htm`.

159. Penchovsky R, Ackermann J. DNA library design for molecular computation. Journal of Computational Biology 2003;10:215–229.

160. D'yachkov AG, Macula AJ, Pogozelski WK, Renz TE, Rykov VV, Torney DC. A weighted insertion-deletion stacked pair thermodynamic metric. In: Ferretti et al. (228) 90–103. SynDCode is available at `http://cluster.ds.geneseo.edu:8080/ParallelDNA/`.

161. Dimitrov RA, Zuker M. Prediction of hybridization and melting for double-stranded nucleic acids. Biophysical Journal 2004;87:215–226.

162. Rose JA, Deaton RJ, Franceschetti DR, Garzon M, Stevens SE Jr. A statistical mechanical treatment of error in the annealing biostep of DNA computation. In: Special program in GECCO-99. 1999 1829–1834.

163. Rose JA, Deaton RJ. The fidelity of annealing-ligation: A theoretical analysis. In: DNA Computing: 6th International Workshop on DNA-Based Computers, DNA 2000 (Leiden Center for Natural Computing: Leiden, The Netherlands), Condon A, Rozenberg G, eds., vol. 2054 of *Lecture Notes in Computer Science*. Springer, 2001 .

164. Rose JA, Deaton RJ, Hayiya M, Suyama A. The fidelity of the tag-antitag system. In: Jonoska and Seeman (223) .

165. Rose JA, Deaton RJ, Hagiya M, Suyama A. An equilibrium analysis of the efficiency of an autonomous molecular computer. Physical Review E 2002;65.

166. Rose JA, Hagiya M, Suyama A. The fidelity of the tag-antitag system II: Reconcilation with the stringency picture. In: Proceedings of the Congress on Evolutionary Computation. 2003 2749–2749. NucleicPark is available at `http://hagi.is.s.u-tokyo.ac.jp/johnrose/` and `http://engronline.ee.memphis.edu/molec/demos.htm`.

167. Rose JA, Deaton RJ, Franceschetti DR, Garzon M, Stevens SE Jr. Hybridization error for DNA mixtures of N species, 1999. `http://engronline.ee.memphis.edu/molec/Misc/ci.pdf`.

168. Rose JA, Suyama A. Physical modeling of biomolecular computers: Models, limitations, and experimental validation. Natural Computing 2004;3:411–426.

169. SantaLucia J Jr, Hicks D. The thermodynamics of DNA structural motifs. Annual Review of Biophysics Biomolecular Structure 2004;33:415–40.

170. Hartemink AJ, Gifford DK. Thermodynamic simulation of deoxyoligonucleotide hybridization for DNA computation. In: Preliminary Proceedings of DNA Based Computers III, DIMACS Workshop 1997 (University of Pennsylvania: Philadelphia, PA), Rubin H, Wood DH, eds. 1997 15–25.

171. Hartemink AJ, Gifford DK, Khodor J. Automated constraint-based nucleotide sequence selection for DNA computation. In: Kari et al. (226) 227–235.

172. Nishikawa A, Yamamura M, Hagiya M. DNA computation simulator based on abstract bases. Soft Computing 2001;5:25–38.

173. Mathews DH, Turner DH. Dynalign: An algorithm for finding the secondary structure common to two RNA sequences. Journal of Molecular Biology 2002;317:191–203.

174. Zuker M. Mfold web server for nucleic acid folding and hybridization prediction. Nucleic Acids Research 2003;31:3406–3415. Mfold is available at http://www.bioinfo.rpi.edu/applications/mfold.

175. Dirks RM, Pierce NA. An algorithm for computing nucleic acid base-pairing probabilities including pseudoknots. Journal of Computational Chemistry 2004;25:1295–1304.

176. Andronescu M, Aguirre-Hernandez R, Condon A, Hoos HH. RNAsoft: a suite of RNA secondary structure prediction and design software tools. Nucleic Acids Research 2003; 31:3416–3422. RNAsoft is available at http://www.rnasoft.ca/.

177. Mathews DH, Disney MD, Childs JL, Schroeder SJ, Zucker M, Turner DH. Incorporating chemical modification constraints into a dynamic programming algorithm for prediction of RNA secondary structure. Proceedings of the National Academy of Sciences of the USA (PNAS) 2004;101:7287–7292. The free energy nearest neighbor parameters are available at http://rna.chem.rochester.edu/, RNAstructure is available at http://128.151.176.70/RNAstructure.html.

178. Hofacker IL. Vienna RNA secondary structure server. Nucleic Acids Research 2003; 31:3429–3431. Vienna Package is available at http://www.tbi.univie.ac.at/~ivo/RNA/.

179. Peyret N, Saro P, SantaLucia J Jr. HyTher server. HyTher Version 1.0 is available at http://ozone2.chem.wayne.edu/.

180. SantaLucia J Jr. A unified view of polymer, dumbbell, and oligonucleotide DNA nearest-neighbor thermodynamics. Proceedings of the National Academy of Sciences of the USA (PNAS) 1998;95:1460–1465.

181. Peyret N, Seneviratne PA, Allawi HT, SantaLucia J Jr. Nearest-neighbor thermodynamics and NMR of DNA sequences with internal A-A, C-C, G-G, and T-T mismatches. Biochemistry 1999;38:3468–3477.

182. Novère NL. MELTING, computing the melting temperature of nucleic acid duplex. Bioinformatics 2001;17:1226–1227. Melting is available at http://www.ebi.ac.uk/~lenov/meltinghome.html.

183. Blake RD, Bizzaro JW, Blake JD, et al. Statistical mechanical simulation of polymeric DNA melting with MELTSIM. Bioinformatics 1999;15:370–375.

184. McDowell JA. MeltWin. MeltWin is available at http://www.meltwin.com/.

185. Flamm C, Fontana W, Hofacker IL, Schuster P. RNA folding at elementary step resolution. RNA 2000;6:325–338. Kinfold is available at `http://www.tbi.univie.ac.at/~xtof/RNA/Kinfold/`.

186. Visual OMP (Oligonucleotide Modeling Platform), DNA Software, Inc. Visual OMP is available at `http://www.dnasoftware.com`.

187. The DNA and Natural Algorithms Group. DNA design toolbox. DNA Design Toolbox is available at `http://www.dna.caltech.edu/DNAdesign/`.

188. Kim D, Shin SY, Lee IH, Zhang BT. NACST/Seq: A sequence design system with multiobjective optimization. In: Hagiya and Ohuchi (227) 242–251.

189. Ruben AJ, Freeland SJ, Landweber LF. PUNCH: An evolutionary algorithm for optimizing bit set selection. In: Jonoska and Seeman (223) 150–160.

190. Bishop M, Macula AJ, Pogozelski WK, Renz TE, Rykov VV. SynDCode: Cooperative DNA code generating software. In: Carbone et al. (231) 391.

191. Pogozelski WK, Bernard MP, Priore SF, Macula AJ. Experimental validation of DNA sequences for DNA computing: Use of a SYBR green assay. In: Carbone et al. (231) 322–331.

192. Yin P, Guo B, Belmore C, et al. Tilesoft: Sequence optimization software for designing DNA secondary structures, 2004. `http://www.cs.duke.edu/~reif/paper/peng/TileSoft/TileSoft.pdf`.

193. Kari L, Kitto R, Thierrin G. Codes, involutions and DNA encodings, 2002.

194. Blain DR, Garzon M, Shin SY, et al. Development, evaluation and benchmarking of simulation software for biomolecule-based computing. Natural Computing 2004;3:427–442.

195. Arita M, Nishikawa A, Hagiya M, Komiya K, Gouzu H, Sakamoto K. Improving sequence design for DNA computing. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000) 2000;875–882.

196. Hoos HH, Stutzle T. Stochastic Local Search: Foundations and Applications. Morgan Kaufmann, 2004.

197. Tulpan DC, Hoos HH, Condon A. Stochastic local search algorithms for DNA word design. In: Hagiya and Ohuchi (227) 229–241.

198. Tulpan DC, Hoos HH. Hybrid randomised neighbourhoods improve stochastic local search for DNA code design. In: Canadian Conference on AI 2003, vol. 2671 of *Lecture Notes in Computer Science*. Springer-Verlag 2003, 2003 418–433.

199. Tulpan D, Andronescu M, Change SB, et al. Thermodynamically based DNA strand design. Nucleic Acids Research 2005;33:4951–4964.

200. Holland JH. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence. Cambridge, MA, USA: MIT Press, 1992.

201. Rechenberg I. Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Information. Freiburg, Germany: Fromman Verlag, 1973.

202. Schwefel HP. Numerical Optimization of Computer Models. New York, NY, USA: John Wiley & Sons, Inc., 1981.

203. Fogel LJ, Owens AJ, Walsh MJ. Artifical Intelligence Through Simulated Evolution. New York, NY: John Wiley & Sons, 1966.

204. Deaton RJ, Murphy RC, Garzon M, Franceschetti DR, Stevens SE Jr. Genetic search of reliable encodings for DNA-based computation. In: First Genetic Programming Conference. Stanford University, 1996 .

205. Deaton RJ, Rose JA. Simulations of statistical mechanical estimates of hybridization error. In: Preliminary Proceedings of the 6th International Workshop on DNA-Based Computers, DNA 2000 (Leiden Center for Natural Computing: Leiden, The Netherlands), Condon A, Rozenberg G, eds. 2000 251–252.

206. Shin SY, Kim DM, Lee IH, Zhang BT. Evolutionary sequence generation for reliable DNA computing. In: Proceedings of the 2002 Congress on Evolutionary Computation (CEC2002), vol. 1. 2002 79–84.

207. Shin SY, Kim DM, Lee IH, Zhang BT. Multiobjective evolutionary algorithms to design error-preventing dna sequences. Tech. Rep. BI-02-003, Biointelligence Lab (BI), School of Computer Science & Engineering, Seoul National University, 2002.

208. Smith WD. DNA computers in vitro and vivo. In: Lipton and Baum (221) 121–185.

209. Tsaftaris SA, Katsaggelos AK, Pappas TN, Papoutsakis ET. DNA-based matching of digital signals. In: International Conference on Acoustics, Speech, and Signal Processing, vol. 5. Montreal, Quebec, Canada, 2004 .

210. Tsaftaris SA, Katsaggelos AK, Pappas TN, Papoutsakis ET. How can DNA computing be applied to digital signal processing? IEEE Signal Processing Magazine 2004;21.

211. Benenson Y, Paz-Elizur T, Adar R, Keinan E, Livneh Z, Shapiro E. Programmable and autonomous computing machine made of biomolecules. Nature 2001;414:430–434.

212. Benenson Y, Adar R, Paz-Elizur T, Livneh Z, Shapiro E. DNA molecule provides a computing machine with both data and fuel. Proceedings of the National Academy of Sciences of the USA (PNAS) 2003;100:2191–2196.

213. Stojanovic MN, Stefanovic D, LaBean T, Yan H. Computing with nucleic acids. In: Bioelectronics: From Theory to Applications, Willner I, Katz E, eds. Wiley-VCH, 2005 .

214. Calude CS, Păun G. Computing with Cells and Atoms. London: Taylor & Francis, 2001.

215. Ehrenfeucht A, Harju T, Petre I, Prescott DM, Rozenberg G. Computation in Living Cells. Berlin: Springer-Verlag, 2004.

216. Păun G, ed. Computing with Bio-Molecules. Singapore: Springer-Verlag, 1998.

217. Rothemund PWK. Folding DNA to create nanoscale shapes and patterns. Nature 2006; 440:297–302.

218. Baron R, Lioubashevski O, Katz E, Niazov T, Willner I. Elementary arithmetic operations by enzymes: A model for metabolic pathway based computing. Angewandte Chemie International Edition 2006;45:1572–1576.

219. Pistol C, Lebeck AR, Dwyer C. Design automation for DNA self-assembled nanostructures. In: Design Automation Conference (DAC). 2006 .

220. Patwardhan J, Johri V, Dwyer C, Lebeck AR. A defect tolerant self-organizing nanoscale simd architecture. In: Proceedings of the Twelth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS XII). 2006 .

221. Lipton RJ, Baum EB, eds. DNA Based Computers, DIMACS Workshop 1995 (Princeton University: Princeton, NJ), vol. 27 of *Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1996.

222. Rubin H, Wood DH, eds. DNA Based Computers III, DIMACS Workshop 1997 (University of Pennsylvania: Philadelphia, PA), vol. 48 of *Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1999.

223. Jonoska N, Seeman NC, eds. DNA Computing: 7th International Workshop on DNA-Based Computers, DNA 2001 (University of South Florida: Tampa, FL), vol. 2340 of *Lecture Notes in Computer Science*. Springer, 2002.

224. Landweber LF, Baum EB, eds. DNA Based Computers II, DIMACS Workshop 1996 (Princeton University: Princeton, NJ), vol. 44 of *Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1999.

225. Winfree E, Gifford DK, eds. DNA Based Computers V, DIMACS Workshop 1999 (MIT: Cambridge, MA), vol. 54 of *Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 2000.

226. Kari L, Rubin H, Wood DH, eds. DNA Based Computers IV, DIMACS Workshop 1998 (University of Pennsylvania: Philadelphia, PA), *Biosystems*, volume 52, issues 1-3. Elsevier, 1999.

227. Hagiya M, Ohuchi A, eds. DNA Computing: 8th International Workshop on DNA-Based Computers, DNA 2002 (Hokkaido University: Sapporo, Japan), vol. 2568 of *Lecture Notes in Computer Science*. Springer, 2003.

228. Ferretti C, Mauri G, Zandron C, eds. DNA Computing: 10th International Workshop on DNA-Based Computers, DNA 2004 (University of Milano-Bicocca: Milan, Italy), vol. 3384 of *Lecture Notes in Computer Science*. Springer, 2005.

229. Chen J, Reif JH, eds. DNA Computing: 9th International Workshop on DNA-Based Computers, DNA 2003 (University of Wisconsin: Madison, WI), vol. 2943 of *Lecture Notes in Computer Science*. Springer, 2004.

230. Ferretti C, Mauri G, Zandron C, eds. Preliminary Proceedings of the 10th International Workshop on DNA-Based Computers, DNA 2004 (University of Milano-Bicocca: Milan, Italy). 2004.

231. Carbone A, Daley M, Kari L, McQuillan I, Pierce N, eds. Preliminary Proceedings of the 11th International Workshop on DNA-Based Computers, DNA 2005 (University of Western Ontario: London, Ontario, Canada). 2005.

232. Peyret N. Prediction of Nucleic Acid Hybridization: Parameters and Algorithms. Ph.D. thesis, Wayne State University, Dept. of Chemistry, 2000.

Tables

Table 8.1: Boolean formulae resulting from the tic-tac-toe game tree.

$$o_1 = i_4$$
$$o_2 = (i_6 \wedge i_7 \wedge \neg i_2) \vee (i_7 \wedge i_9 \wedge \neg i_1) \vee (i_8 \wedge i_9 \wedge \neg i_1)$$
$$o_3 = (i_1 \wedge i_6) \vee (i_4 \wedge i_9)$$
$$o_4 = i_1$$
$$o_5 = 1$$
$$o_6 = (i_1 \wedge i_2 \wedge \neg i_6) \vee (i_1 \wedge i_3 \wedge \neg i_6) \vee (i_1 \wedge i_7 \wedge \neg i_6) \vee (i_1 \wedge i_8 \wedge \neg i_6) \vee (i_1 \wedge i_9 \wedge \neg i_6)$$
$$o_7 = (i_2 \wedge i_6 \wedge \neg i_7) \vee (i_6 \wedge i_8 \wedge \neg i_7) \vee (i_6 \wedge i_9 \wedge \neg i_7) \vee (i_9 \wedge i_2 \wedge \neg i_1)$$
$$o_8 = i_9 \wedge i_7 \wedge \neg i_4$$
$$o_9 = (i_7 \wedge i_8 \wedge \neg i_4) \vee (i_4 \wedge i_2 \wedge \neg i_9) \vee (i_4 \wedge i_3 \wedge \neg i_9) \vee (i_4 \wedge i_6 \wedge \neg i_9) \vee (i_4 \wedge i_7 \wedge \neg i_9) \vee (i_4 \wedge i_8 \wedge \neg i_9)$$

Figures

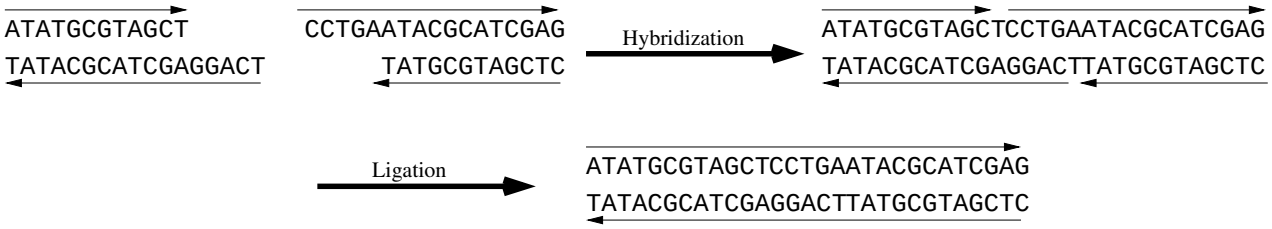


Figure 8.1: Hybridization of the single-stranded sticky ends extending from double-stranded DNA molecules. After the base pair bonding occurs in hybridization, the backbones of the two dsDNA molecules may be joined by ligation.

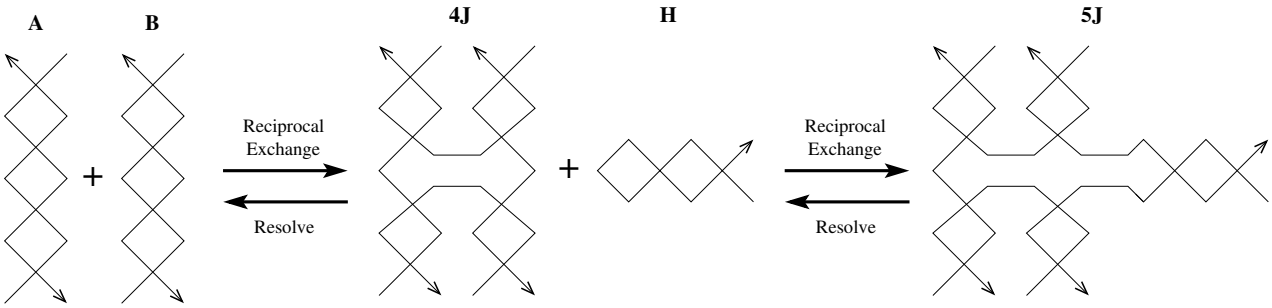


Figure 8.2: The opposite-polarity dsDNA molecules A and B undergo reciprocal exchange to form the four-arm branched junction 4J. The 4-arm junction 4J then undergoes reciprocal exchange with the hairpin molecule H to form the 5-arm branched junction molecule 5J.

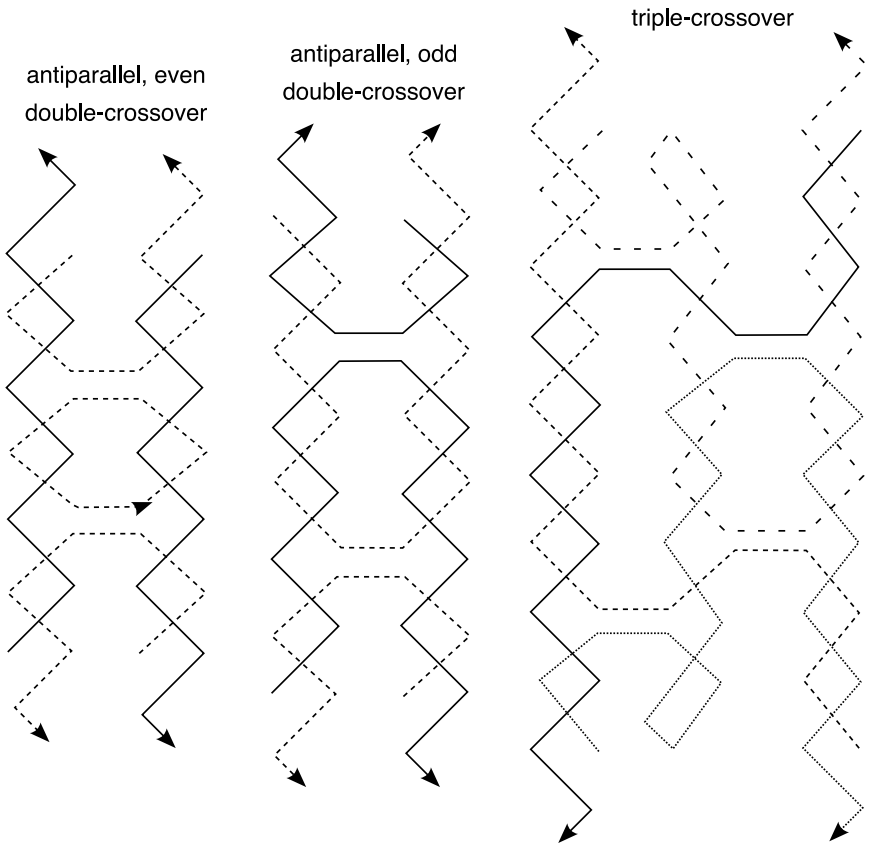


Figure 8.3: Two antiparallel double-crossover DNA molecules, and a triple-crossover molecule. The even and odd labels on the double-crossover molecules refer to the number of helical half-turns between the two crossovers (two in the left molecule, three in the middle molecule). The double-crossover molecules are formed when two crossovers occur between two double-stranded DNA molecules, while the triple-crossover molecule is formed when four crossovers occur between three double-stranded DNA molecules. The different line styles represent different contiguous single DNA strands in the new molecule. The four extended strands on each molecule are "sticky ends" that can be used to connect DNA tiles together.

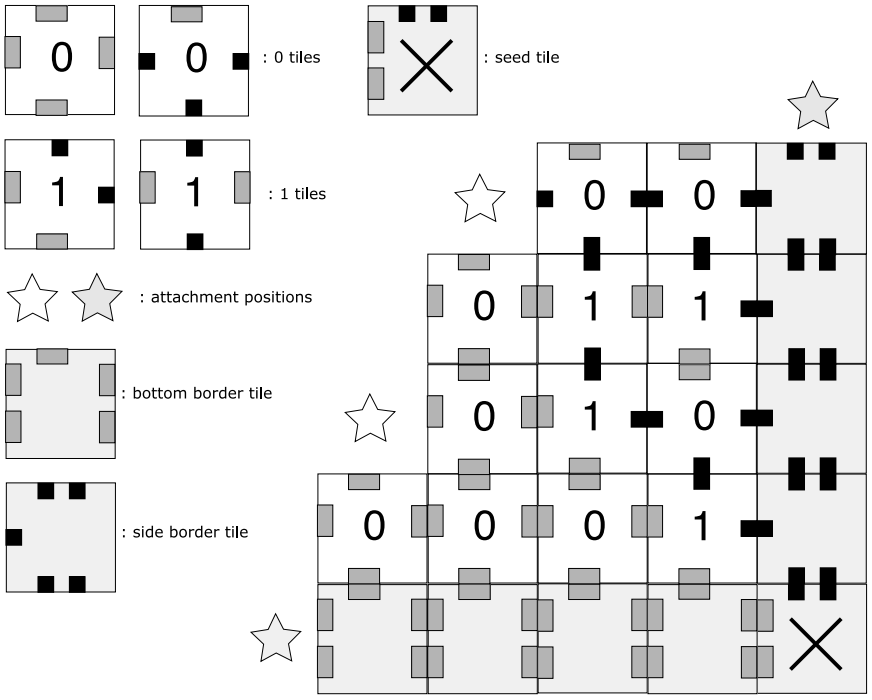


Figure 8.4: A binary counter in the process of self assembly. The seed tile starts off the assembly. The right side and bottom border tiles connect to each other with double bonds, while all the other tiles connect with single bonds. A tile needs two single bonds (or one double bond) to form a stable attachment to the structure; the marked attachment positions show where a tile can form a stable attachment.

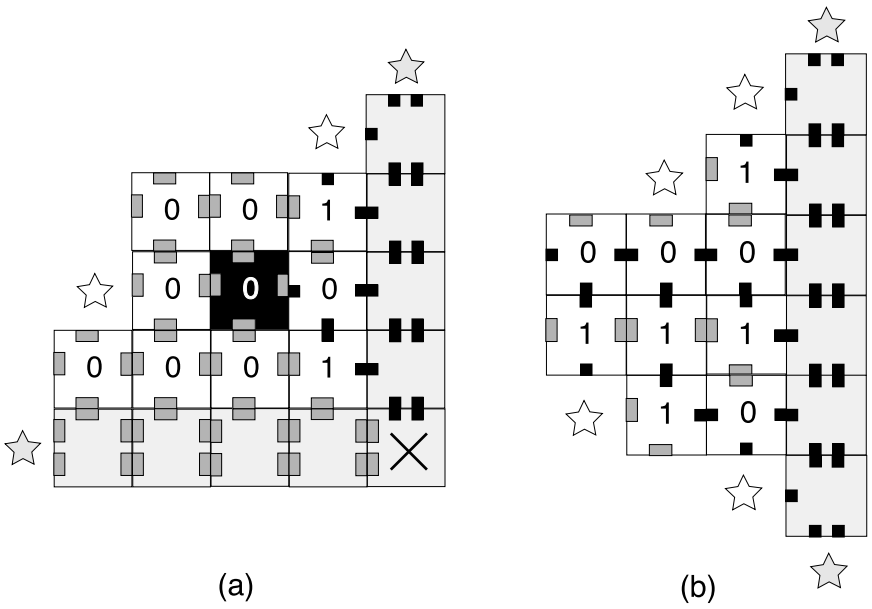


Figure 8.5: Errors present in two binary counters in the process of self-assembly. In (a), the tile highlighted in black is *mismatched*, but has been locked into place by other tiles binding correctly around it. Hence, in this case, our counter counts to one, then zero, then one again; obviously, a similar error can be arbitrarily serious, destroying the counter's count. In (b), we see two types of errors. The boundary tiles have formed without growing off of a seed tile (corner tile); this is a *nucleation error*. Also, although there are no mismatches, the rule tiles have begun counting at 8 (or more, depending on whether more ones or zeros bind to the frontier) and are continuing forward and backward. This is because they started assembling on a facet (edge) rather than in the corner as in (a). This constitutes a *facet error*.

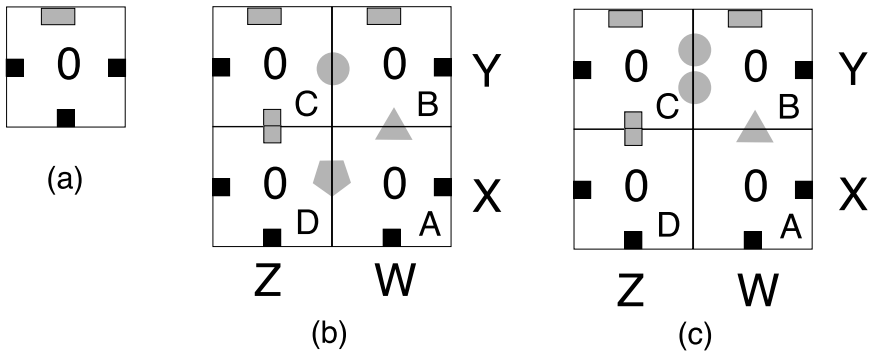


Figure 8.6: A tile, (a), and 2x2 proofreading tile sets representing it, (b) and (c). (b) is the simple redundant representation. The assembly is growing from right to left and top to bottom (as in the binary counter example). Note that all four tiles are connected to each other with unique types of single-bonds. (c) is the improved, "snake" proofreading tile set representation, so named because its formation snakes back upon itself. We can see that the snake tileset greatly decreases the chance of a facet nucleation error (when a tile binds to some facet instead of at a corner, and is then locked into place by another tile). Recall that a tile must be attached with two bonds (or one double-bond) to be a stable part of the structure. If tile A in (b) formed a single-bond with Z, for example, it could be locked in place by tile D binding (in a stable, two-bond manner) to its left, and so the error propagates to the left after only one single-bond facet nucleation (A binding to Z). However, with the snake tileset, there can be no bond between A and D. In order for the block in (c) to grow, A must bind to B, which binds (with a stable, double-bond) to C, which then binds to D. Thus, the set in (c) would require two undesired single-bonds in very close proximity (namely, A to Z and B to A) before only double-bonds are required to lock the error in place (C to B, etc.). The probability of this happening is very small.

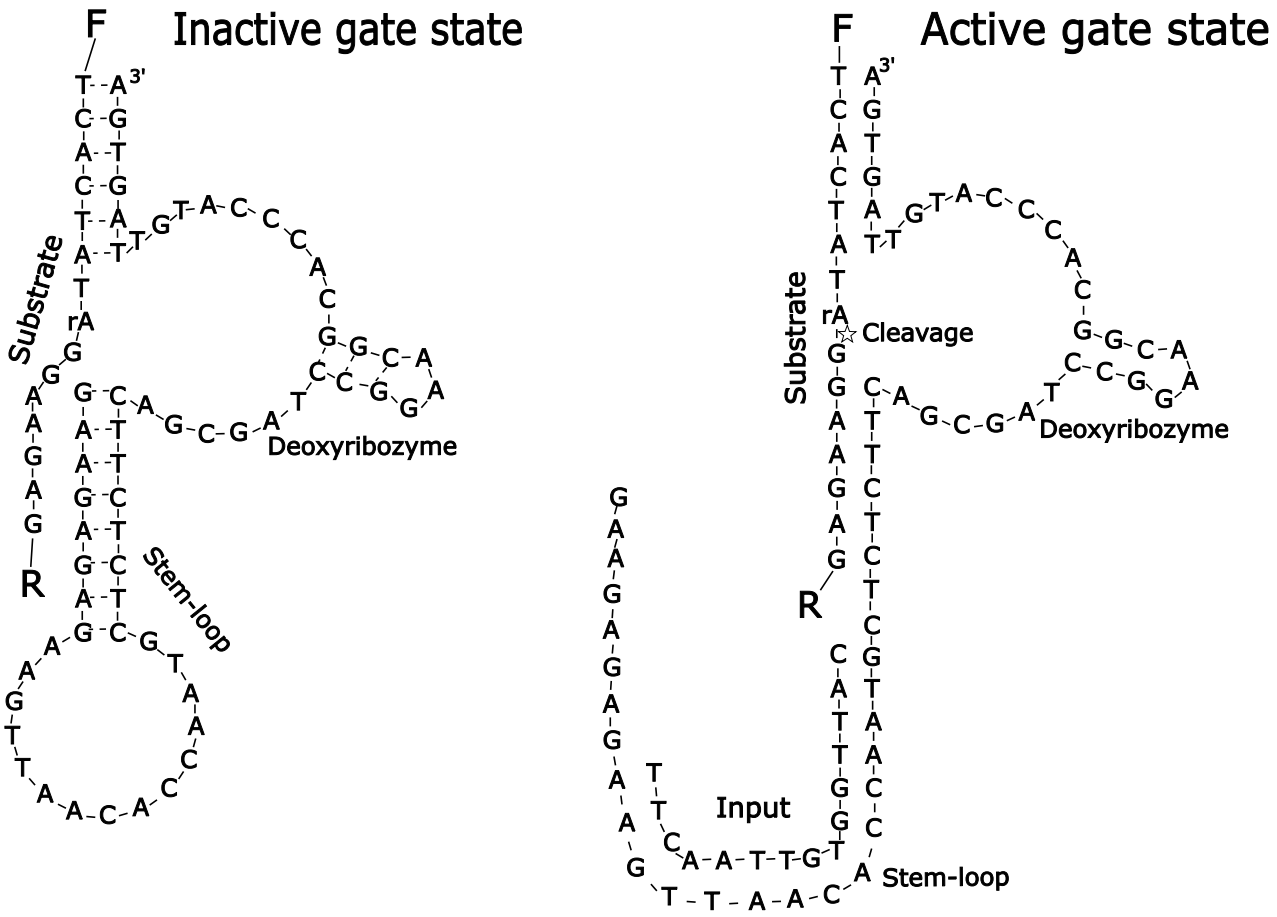


Figure 8.7: A YES gate, in which an "input" oligonucleotide activates a deoxyribozyme by opening an inhibitory stem.

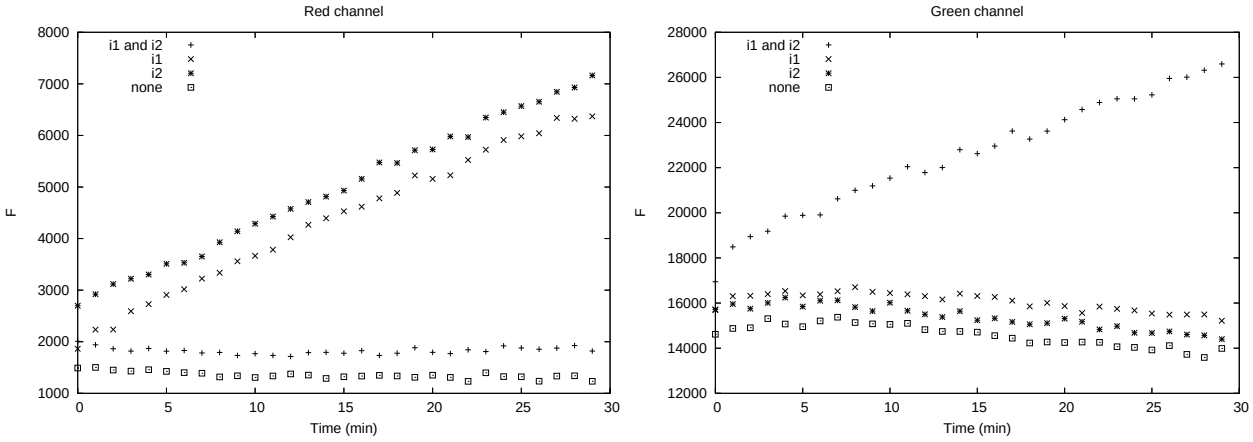


Figure 8.8: Observed fluorescence change in a half-adder deoxyribozyme logic circuit: the red tetramethylrhodamine channel is shown on the left; the green fluorescein channel is shown on the right.

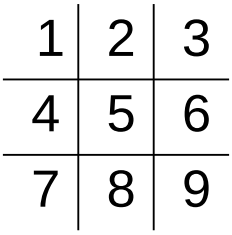
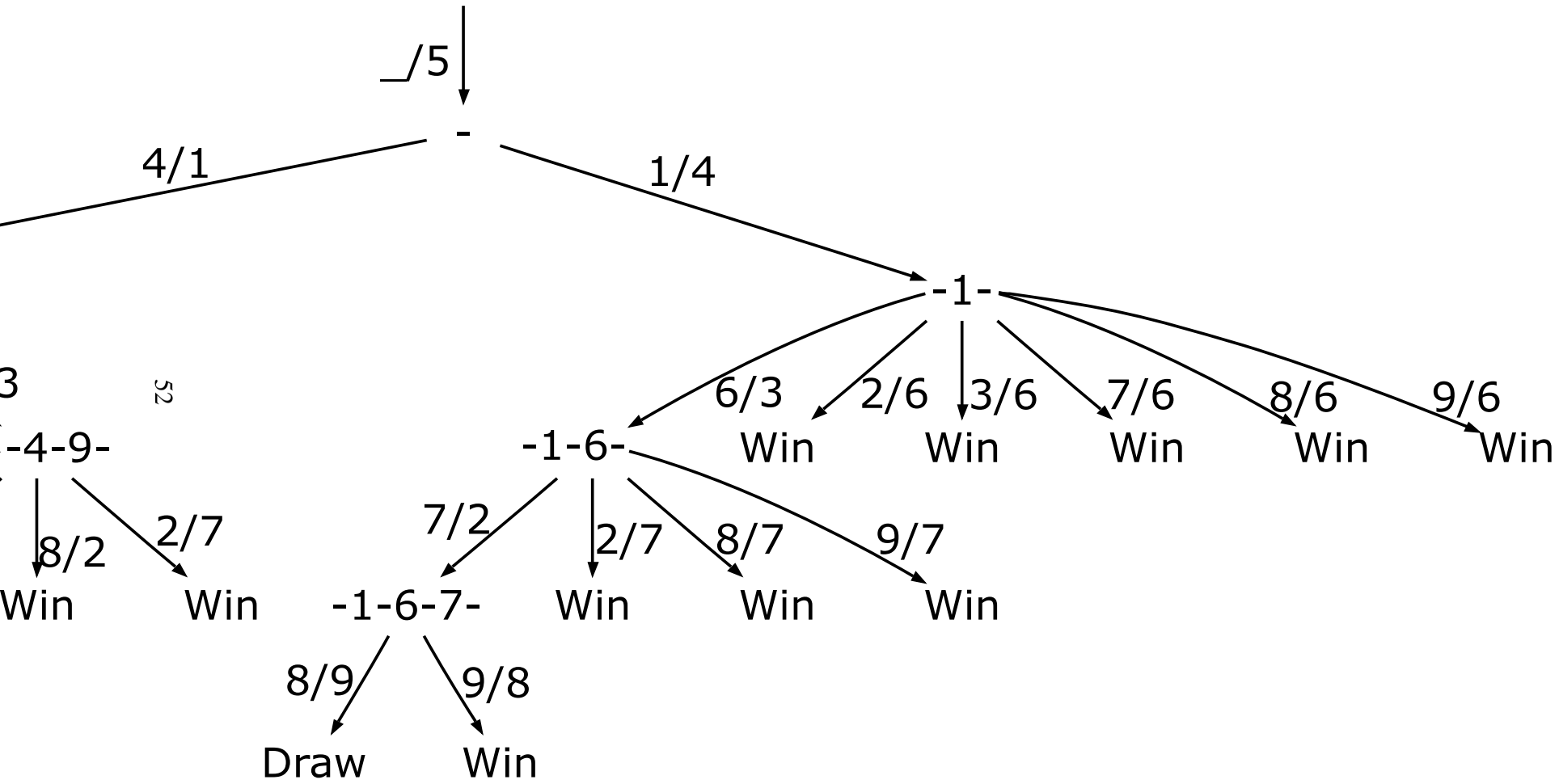


Figure 8.9: The tic-tac-toe game board.


_/5
-
4/1
1/4
-1-
6/3
2/6
3/6
7/6
8/6
9/6
Win
Win
Win
Win
Win
-1-6-
7/2
2/7
8/7
9/7
Win
Win
Win
-1-6-7-
8/9
9/8
Draw
Win
-4-9-
8/2
2/7
Win
Win

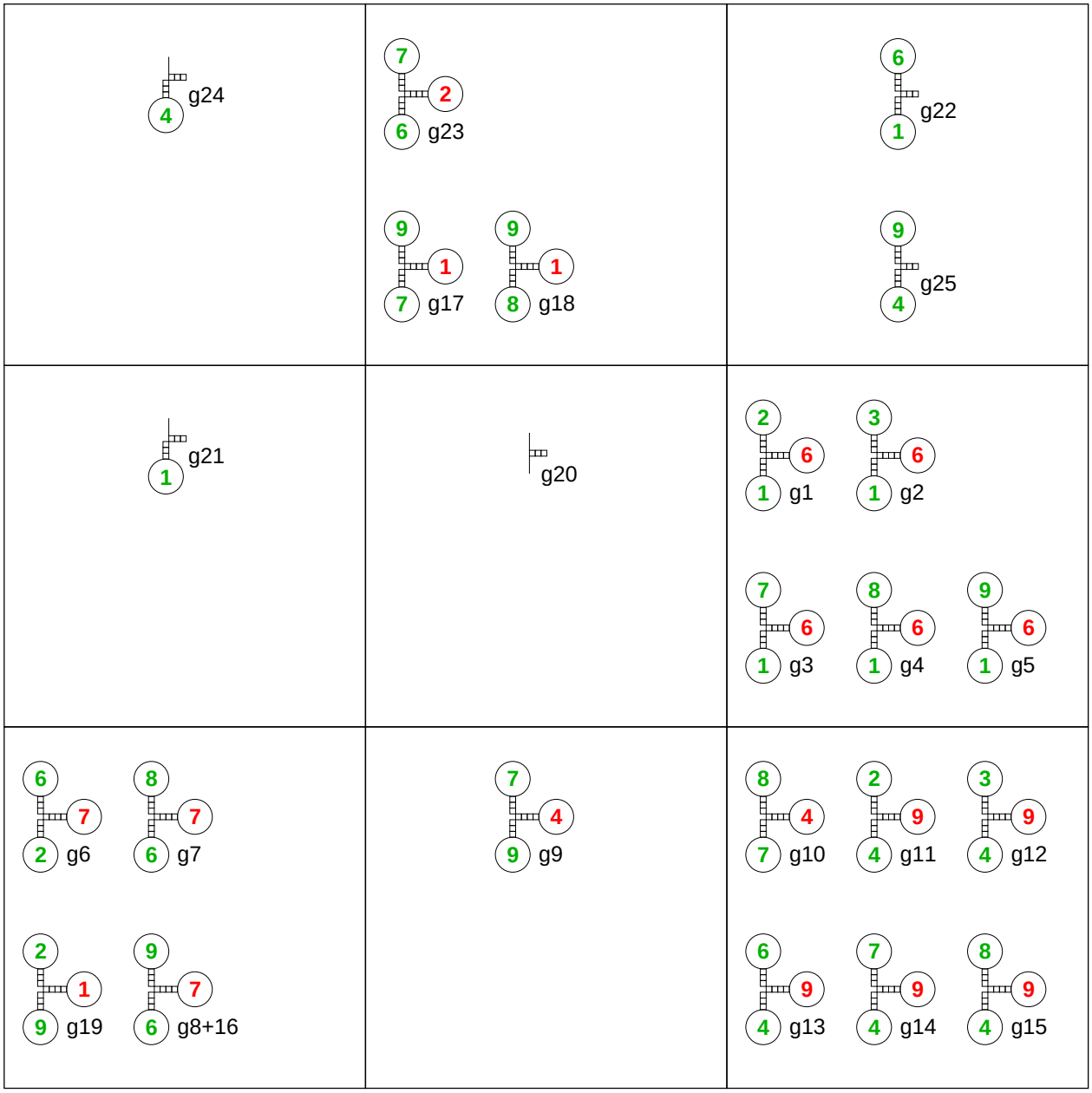


Figure 8.11: Realizing a tic-tac-toe automaton using deoxyribozyme logic. The center well contains a consitutively active deoxyribozyme. Each of the eight remaining wells contains a number of deoxyribozyme logic gates as indicated.

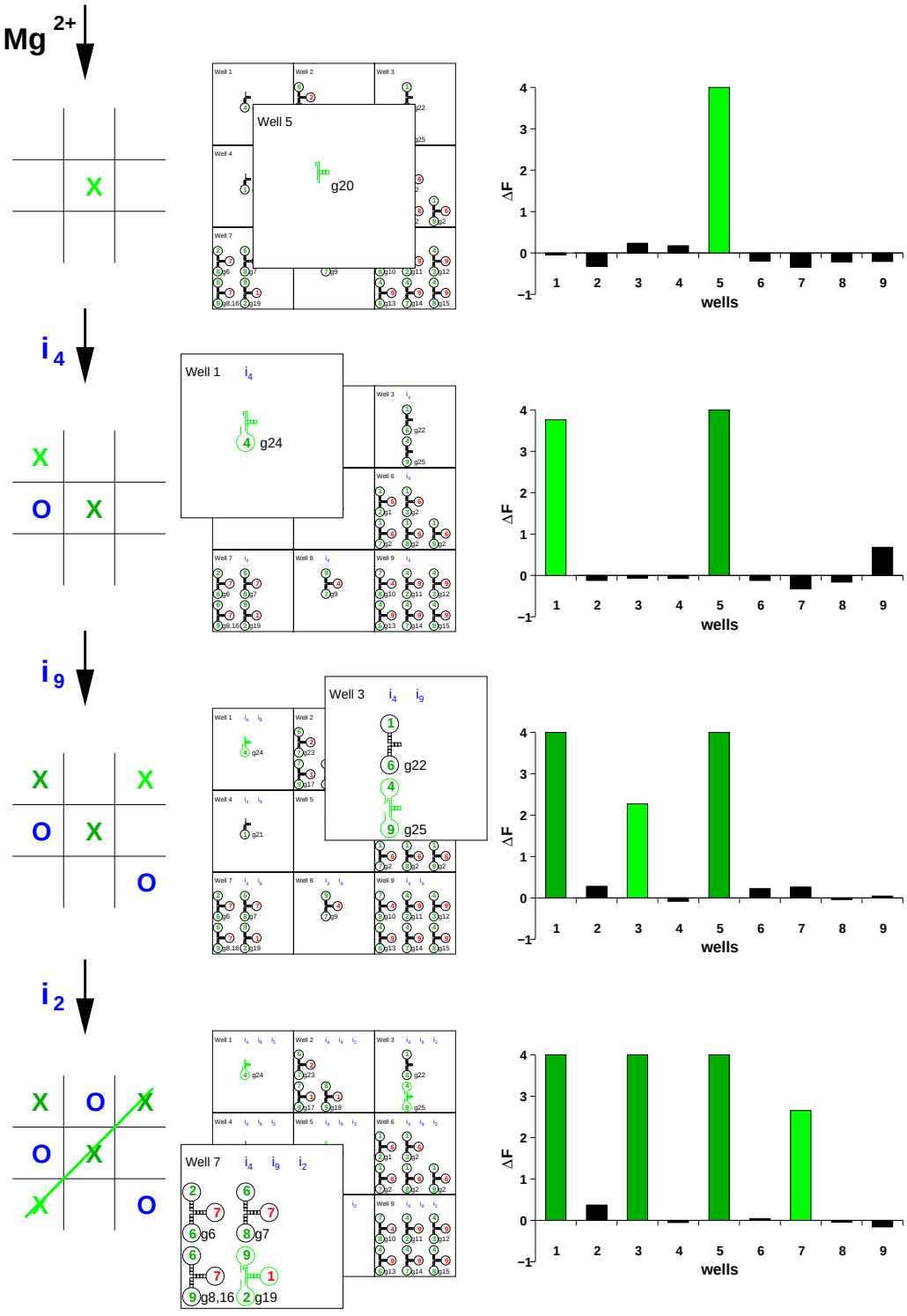


Figure 8.12: A game of tic-tac-toe. See main text for description.

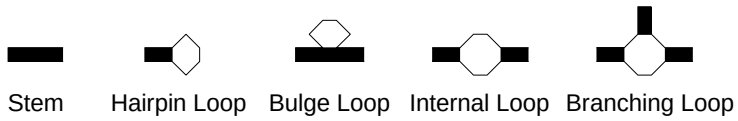


Figure 8.13: DNA loops. Solid areas represent double stranded sections. Lines represent single stranded sections.

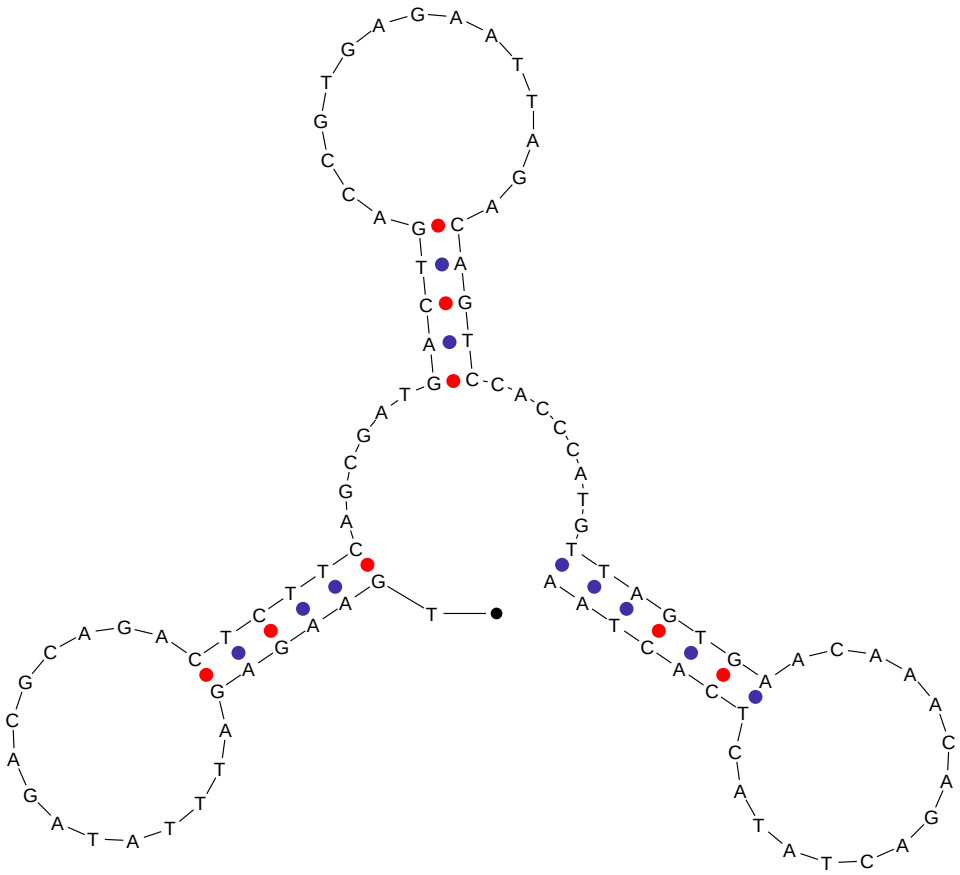


Figure 8.14: Example of secondary structure in Stojanovic and Stefanovic's DNA automaton (112) as computed by MFold (174,180,232) using 140 mM Na^{+}, 2 mM Mg^{2+}, and 25°C. The strand has three hairpin loops, which is the desired secondary structure. ΔG is -12.3 kcal/mol.