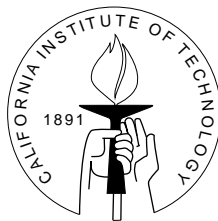# Algorithmic Self-Assembly of DNA

Thesis by
Erik Winfree

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology
Pasadena, California

1998
(Submitted May 19, 1998)

Acknowledgements

This thesis reports an unusual and unexpected journey through intellectual territory entirely new to me. Who is prepared for such journeys? I was not. Thus my debt is great to those who have helped me along the way, without whose help I would have been completely lost and uninspired.

There is no way to give sufficient thanks to my advisor, John Hopfield. His encouragement for me to get my feet wet, and his advice cutting to the bone of each issue, have been invaluable. John's policy has been, in his own words, to give his students "enough rope to hang themselves with." But he knows full well that a lot of rope is needed to weave macrame. Perhaps the only possible repayment is in kind: to maintain a high standard of integrity, and when my turn comes, to provide a nurturing environment for other young minds.

Len Adleman and Ned Seeman have each been mentors during my thesis work. In many ways, my research can be seen as the direct offspring of their work, combining the notion of using DNA for computation with the ability to design DNA structures with artificial topology. Both Len and Ned have provided encouragement, support, and valuable feedback throughout this project.

I would like additionally to thank Ned Seeman and Xiaoping Yang for teaching me how to do laboratory experiments with DNA. Xiaoping's patient and careful tutoring provided me with a solid first step toward becoming a competent experimenter. John Abelson generously made me welcome to continue my experimenting at Caltech in a friendly and expert environment; I am deeply grateful to him and to all the members of his laboratory without whose help I could have done little. For teaching me to use the atomic force microscope, I thank Anca Segall, Bob Moision, and Ely Rabani; with their encouragement and help I saw the first exciting images of DNA lattices. Likewise, Rob Rossi's friendly and spirited management of the Caltech SPM room made doing science there a breeze.

I am grateful for the encouragement and feedback of my thesis committee: John Abelson, Yaser Abu-Mostafa, Len Adleman, John Baldeschwieler, Al Barr, and John Hopfield.

Perhaps most influential have been the people in my everyday life. All the members of the Hopfield group – Carlos Brody, Dawei Dong, Maneesh Sahani, Marcus Mitchell, Sam Roweis, Sanjoy Majahan, Tom Annau, and Unni Unnikrishnan – have been wonderful to be with, both on and off duty. Laura Rodriguez has been our guardian angel. Paul Rothemund has been a steady and stimulating companion throughout this DNA adventure, and it is fair to say that this project would never have been born without him. Matt Cook has been a source of intellectual inspiration and joy for many years. I have learned so much from you all. I will miss you all when we are apart; let those times be brief!

I must add that I feel very fortunate to have come to a place – Caltech's CNS program – where as a naive mathematician and computer scientist, I can build an autonomous robot car, do electrophysiology on the brain of rat and listen to the cries of the jungle therein, and weave macrame out of life's genetic molecules. There's nothing else like it. Therefore I also want to thank Caltech and all the people that compose it's unique community. Nowhere have I felt intellectually more at home, and among people like myself.

I am grateful for stimulating discussions with many scientists in the wider community, including John Reif, Dan Abrahams-Gessel, Tony Eng, Natasha Jonoska, James Wetmur, and many others. I thank Takashi Yokomori for inviting me and hosting me on a once-in-a-lifetime trip to Japan, where I was able to share ideas with a great number of people who think about DNA computing; I would like to name in particular Masami Hagiya, Shigeyuki Yokoyama, Masanori Arita,

Abstract

How can molecules compute? In his early studies of reversible computation, Bennett imagined an enzymatic Turing Machine which modified a hetero-polymer (such as DNA) to perform computation with asymptotically low energy expenditures. Adleman's recent experimental demonstration of a DNA computation, using an entirely different approach, has led to a wealth of ideas for how to build DNA-based computers in the laboratory, whose energy efficiency, information density, and parallelism may have potential to surpass conventional electronic computers for some purposes. In this thesis, I examine one mechanism used in all designs for DNA-based computer – the self-assembly of DNA by hybridization and formation of the double helix – and show that this mechanism alone in theory can perform universal computation. To do so, I borrow an important result in the mathematical theory of tiling: Wang showed how jigsaw-shaped tiles can be designed to simulate the operation of any Turing Machine. I propose constructing molecular Wang tiles using the branched DNA constructions of Seeman, thereby producing self-assembled and algorithmically patterned two-dimensional lattices of DNA. Simulations of plausible self-assembly kinetics suggest that low error rates can be obtained near the melting temperature of the lattice; under these conditions, self-assembly is performing reversible computation with asymptotically low energy expenditures. Thus encouraged, I have begun an experimental investigation of algorithmic self-assembly. A competition experiment suggests that an individual logical step can proceed correctly by self-assembly, while a companion experiment demonstrates that unpatterned two dimensional lattices of DNA will self-assemble and can be visualized. We have reason to hope, therefore, that this experimental system will prove fruitful for investigating issues in the physics of computation by self-assembly. It may also lead to interesting new materials.

Contents

Chapter 1 Contributions

1.1 Introduction to DNA-Based Computation

How can molecules be used to compute? The ground-breaking work of Adleman (1994) showed, in analogy with *in vitro* selection techniques in combinatorial chemistry, how DNA sequences can encode mathematical information and how simple sequences of standard molecular biology experiments can be used to isolate the DNA which encodes the answer to a difficult mathematical problem. I review this work, and its extensions by Lipton (1995). I place a complexity-theoretic limit on what mathematical information can be isolated by n steps of affinity separation alone and by n steps of affinity separation in combination with PCR amplification. This emphasizes the contribution of Boneh et al. (1996a), who show a technique that uses affinity separation in combination with ligation to overcome the limit.

Hagiya et al. (in press) proposed a novel experimental technique which promises to simplify the selection process for DNA-base computation. In their technique, a single chemical reaction based on PCR can perform a sequence of logical operations autonomously. I present a new analysis of the computational power of this technique, highlighting the role of the combinatorial generation of structured sets of DNA strands. I show how to solve the Formula Satisfiability, Independent Set, and Hamiltonian Path problems using this technique, and I propose a novel extension of the technique to solve the Circuit Satisfiability problem.

1.2 Models of Computation by Self-Assembly

Since Adleman's original paper, every proposal for DNA-based computation has made use of the sequence-specific hybridization of Watson-Crick complementary oligonucleotides. Most applications have been very straightforward, and the the most sophisticated use of this self-assembly is still Adleman's original technique for creating duplex DNA representing paths through a graph. However, much more elaborate DNA constructs are possible, as epitomized by Seeman's extensive experimental research in DNA nanoconstructions: in addition to duplex DNA, hairpins, n-arm junctions, and double-crossover molecules are all possible. Using this expanded vocabulary, what computations can be done with self-assembly alone? To answer this question, I use the framework of formal language theory to develop a model of DNA self-assembly in which such questions can be rigorously answered. The surprising result is that in the two-dimensional case the self-assembly model is Turing-universal, and that natural restrictions of the model reproduce the Chomsky Hierarchy of language families. These restrictions relate to the types of DNA building-blocks used, and the form of their arrangement into larger structures: the self-assembly of linear duplex DNA into linear polymers produces regular languages; the self-assembly of duplexes, hairpins and 3-arm junctions into dendrimers produces context-free languages; and the self-assembly of double-crossover molecules into two-dimensional lattices achieves Turing-universality, producing recursively enumerable languages.

To make analysis possible, the theoretical models had to make several simplifying assumptions that would not strictly hold in the real world. How severe is this inaccuracy, and is it plausible to design DNA molecules whose real behavior mimics that of the model? The thermodynamics and kinetics of DNA hybridization have been extensively studied, providing a solid foundation for a

quantitative plausibility argument. To apply this knowledge to the two-dimensional case, one must know whether multiple binding domains are cooperative. Using the assumption that binding energies are additive, I have developed equations for the kinetics of the two-dimensional self-assembly process – akin to 2D crystal growth – and implemented the equations in a computer simulation. The simulation results suggest error-free growth occurs when a system with low concentrations of the DNA monomers is held near the melting temperature of the DNA lattice.

1.3 Experiments with Self-Assembly

Can the proposed models be implemented experimentally? The simulations made use of two assumptions that had no direct experimental support: (1) that the envisioned two-dimensional lattices can be made, independently of whether any computation can be embedded in them, and (2) that the four binding domains in double-crossover molecule act cooperatively in a growing two-dimensional lattice. I therefore performed experimental tests of these two hypotheses. Each experiment involved three stages: the design of sequences for DNA oligonucleotides composing the desired building blocks, the synthesis and self-assembly of those oligonucleotides into building blocks and the subsequent self-assembly of the building blocks into larger structures, and the experimental analysis and characterization of the resulting structures.

To assist in the design of sequences for the coming experiments, which involve many tens of oligonucleotides and thousands of nucleotide positions, I developed software tools for evaluating sequences according to various heuristic criteria and for automatically optimizing to find improved sequences according to the criteria.

Question (2) was approached first. In work with collaborators Seeman and Yang, a 150-K Dalton molecular system was designed to model the binding site in a growing two-dimensional lattice of double-crossover molecules. As envisioned for the lattice, the binding site consisted of two single-stranded DNA binding domains available for hybridization, separated by 20 nm. Cooperativity was tested by competition of binding between two molecules. The target molecule had perfect complementarity to both binding domains, while the ersatz molecule had perfect complementarity to one binding domain but 50% mismatches in the other domain. Even in the presence of a 64-fold excess of the ersatz molecule, the target molecule was preferred in the binding site, indicating cooperativity.

Question (1) was then addressed. I designed a system of two double-crossover molecules that can self-assemble into a two-dimensional lattice. The double-crossover molecules and the resulting lattice were characterized by gel electrophoresis and visualized by atomic force microscopy. Attaching a bulky DNA "arm" to just one of the double-crossover molecules produced stripes with the expected period in the atomic force microscope images, confirming the correct lattice structure of the self-assembled crystal. A similar system was investigated in Seeman's lab.

These two properties, that double-crossover molecules can self-assemble into a two-dimensional crystal and that the two binding domains at binding sites during lattice growth are cooperative, are the key ingredients for a real implementation of the Turing-universal model of computation by self-assembly of DNA.

1.4 Publication List

This thesis contains material from several conference publications and one journal article. I was the first author on all papers, and the writing is primarily my own. The creative ideas and the

actual labor for all the results presented here are due primarily to me, except where explicitly noted. Some of the text and figures in this thesis come directly from those articles, although most of it has undergone revision, and occasionally correction, for incorporation into this thesis. I am solely responsible for any mistakes herein.

Chapter 2 uses material from:

> Erik Winfree, "Complexity of Restricted and Unrestricted Models of Molecular Computation" (Winfree 1996a).

> Erik Winfree, "Whiplash PCR for $O(1)$ Computing" (Winfree in press b).

Chapter 3 uses material from:

> Erik Winfree, "On the Computational Power of DNA Annealing and Ligation" (Winfree 1996b). The ideas in this paper, although due to me, were heavily influenced by discussions with Seeman, in particular with respect to the choice of the double-crossover molecule to implement the tiles.

> Erik Winfree, Xiaoping Yang, Nadrian C. Seeman, "Universal Computation via Self-Assembly: Some Theory and Experiments" (Winfree et al. in press). Chapter 3 discusses the theoretical results in this paper, which are due entirely to me.

> Erik Winfree, "Simulations of Computation by Self-Assembly" (Winfree in press a).

Chapter 4 uses material from:

> Erik Winfree, Xiaoping Yang, Nadrian C. Seeman, "Universal Computation via Self-Assembly: Some Theory and Experiments" (Winfree et al. in press). Chapter 4 discusses the experiments reported in this paper. Seeman outlined the experiments and designed the DNA sequences. Yang designed the experimental details and supervised my execution of the laboratory techniques for initial experiments. I designed and carried out all further experiments myself.

> Erik Winfree, Furong Liu, Lisa A. Wenzler, Nadrian C. Seeman, "Design and Self-Assembly of Two-Dimensional DNA Crystals" (Winfree et al. 1998). This paper describes two parallel experimental investigations of an idea derived from Winfree (1996b); the creative ideas in this paper are due to Seeman and myself. The experiments on DAE molecules were designed and carried out by Liu, Wenzler, and Seeman; the experiments on DAO molecules were designed and carried out by myself. Chapter 4 of this thesis presents only the results on DAO molecules.

1.5   Support

Chapter 2   Introduction to DNA-Based Computation

2.1   Why Compute with Molecules, and How?

All computers are physical objects, made from atoms and molecules, and are governed by the laws of physics. For most purposes, this fact can readily be ignored, and computers can be analyzed at a purely logical level. However, as Moore's Law plays out and computers are built out of ever smaller devices, the atomic, molecular, and quantum nature of those devices becomes ever more important – as do the fundamental physical limits of computation, such as those imposed by reversibility, heat generation, and thermal noise. The best architectures for computers built at these scales may be very different from the ones we are familiar with now.

An examination of molecular biology provides important hints for information processing by molecules. Some 5×10^9 bits of information are stored in the human genome, in the nucleus of every living cell in your body, at a density near 1 bit per nm^3. A single cell contains on the order of 10^9 active macromolecules (proteins, enzymes, polynucleotides,...) acting in parallel to control the functions of the cell, despite thermal noise and the randomness inherent in diffusible elements. However, it is not immediately clear *what* a cell is "computing," or how one could make use of molecular mechanisms like those in the cell for computing.

Still, to a computer scientist, the mechanisms *look* like computational primitives, and one of the central themes of computer science has been that just about any grab bag of primitives is theoretically sufficient for building a Turing-universal machine. We will see that the molecular biology grab bag also suffices.

2.2   Computing Inverse Sets with DNA

Abstract[1] In Adleman's paradigm for solving combinatorial problems with DNA, the problem to be solved is encoded as a sequence of experiments to be performed upon a combinatorial library of DNA. We would like this sequence of experiments to be as brief as possible. Thus we examine the expressive power of different experimental paradigms. We begin with the formal models of Lipton (1996b) and Adleman (1996), which make focused use of affinity purification. The use of PCR was suggested in Lipton (1996b) to expand the range of feasible computations, resulting in a second model. By giving a precise characterization of these two models in terms of recognized computational complexity classes, namely branching programs (BP) and nondeterministic branching programs (NBP) respectively, we show that PCR only incrementally increases the computational power. However, the use of ligation, introduced by Boneh et al. (1996b), results in a third model which does significantly increase the computational power.

---

[1]Results in this section also appeared in Winfree (1996a). Thanks to Sam Roweis for stimulating discussions and to Jehoshua Bruck for pointing me to previous literature on branching programs.

Current interest in using DNA to compute was spurred by Adleman (1994), who brought DNA-based computers out of the realm of theory and into experimental reality. Adleman's key insight was to avoid trying to use each DNA strand as a basis for a complex processor, and instead to use a vast collection of simple DNA strands to collectively perform a single computation. In his solution to the Hamiltonian Path Problem (HPP), he used standard molecular biology techniques to perform two types of logical manipulation of the DNA. First, he used sequence-directed polymerization of oligonucleotides to generate a combinatorial set of DNA strands[2] representing paths through a graph G with n vertices. The sequences of the resulting strands encode the vertices visited by each respective strand. Second, Adleman used a series of PCR reactions, gel electrophoresis experiments, and affinity separations to get rid off strands in the combinatorial library that surely *didn't* represent the correct answer. Paths which weren't length n were removed, then paths which omitted vertex 1 were removed, and so on until paths which omitted vertex n were removed. The remaining DNA represented valid answers to the problem.

This approach was quickly generalized by Lipton (1995), who noted that it is sufficient to always start with the maximally diverse combinatorial library representing all 2^n binary bitstrings, and to filter that set down to the desired solution. In particular, he showed how to solve the formula satisfiability problem (FSAT): given a Boolean formula with s terms in n variables, Lipton showed how a series of $O(s)$ affinity separation steps could be performed to find DNA which encodes values to those variables which make the formula true. Because a minute 100 μl solution can contain 10^{15} strands of DNA and a single laboratory operation processes all those strands in parallel, at first glance it appears that for $2^n < 10^{15} \approx 2^{50}$, size n problems can be solved in $O(n)$ steps. As FSAT is among the hardest of the hard problems, this generated excitement.

Considered generally, molecular computation as introduced by Adleman (1994) provides a new approach to solving combinatorial inverse problems, where we are interested in computing $f^{-1}(1)$ where $f(\mathbf{x})$ is a boolean function of n-bit strings $\mathbf{x}$. Instances of NP-complete problems can be expressed in this form; for example, in 3-SAT we ask if $f^{-1}(1)$ is non-empty for f given as a 3-CNF expression. Adleman's technique involves using individual DNA strands to represent potential answer bit-strings $\mathbf{x}$, then operating on a test tube containing all possible answers to separate those which satisfy f from those which don't. In many instances, the number of sorting operations required is a low-order polynomial in n, suggesting that – given exponential space to store the DNA – hard combinatorial problems can be solved efficiently with this technique. Because the bounded resource of space to store the DNA is so critical, in this discussion we will only consider using $O(2^n)$ strands. Using substantially more DNA, e.g. to search over additional non-deterministic variables, is considered "cheating". In other words, the question is, "Given a fixed amount of DNA, what functions can we easily solve?"

It was not immediately clear, however, what class of boolean functions f could be efficiently inverted. In a clarifying paper, Lipton (1996b) showed that if f can be represented as a size L formula of AND-OR-NOT (AON) operations, then f can be inverted using $2L$ molecular steps using affinity purification only. Lipton suggested further that the use of PCR to duplicate the contents of a test tube would allow an even greater class of functions to be inverted using molecular computation. In this note we follow his program and characterize exactly to what extent PCR helps, in terms of known complexity classes.

As individual steps can take on the order of 15 minutes to an hour, small differences in complexity quickly make the difference between feasible and infeasible experiments. Thus it is of

---

[2]The terms *strand, oligonucleotide, oligo,* and *polynucleotide* all refer single-stranded DNA molecules, and they are all roughly interchangeable. However, "oligo" means "a few" and thus refers to short strands – a few tens of nucleotides. A combinatorial set of DNA strands is called a *combinatorial library* for short.

importance to characterize the complexity of these models of molecular computation as carefully as possible. Classes such as "polynomial-size" are too rough to be really useful – we really want to know exactly what polynomial it is.

After defining the two models of molecular computation, we will demonstrate their correspondence with branching programs, and conclude with a few implications of the correspondence.

2.2.1 Abstract Models of Molecular Computation

We use the models described in Lipton (1996b) and Adleman (1996), and use similar notation. These models assume perfect performance of each operation, although in practice the molecular biology techniques are known to be somewhat unreliable. Initial comments on this aspect of the models, the origin of the names *restricted model* and *unrestricted model*, and other practical matters, can be found in Adleman (1996).

The Restricted Model:

A *test tube* is a set of molecules of DNA encoding assignments of values to variables $x_1 \ldots x_n$. Each assignment, e.g., $x_{17} = 1$, is encoded using a unique DNA sequence, sufficiently dissimilar from encodings of other assignments. Each DNA strands is simply the concatenation of all assignment encodings. We operate on test tubes as follows:

- *Separate[i].* Given a tube T and an index[3] i, produce two tubes $+(T, i)$ and $-(T, i)$, where $+(T, i)$ contains all strings where bit i is set, and $-(T, S)$ contains all strings where bit i is cleared. Tube T is destroyed.

- *Merge.* Given tubes T_a and T_b, pour T_b into T_a thereby making $T_a \leftarrow T_a \cup T_b$. Tube T_b is destroyed.

Separate is implemented using affinity separation based on the presence of the appropriate DNA sequence (Adleman 1994), and the implementation of *merge* is obvious. At the end of the computation[4], when we presumably have a single test tube containing all strings in $f^{-1}(1)$, we can use the following operation to sequence the strings $\mathbf{x}$ in the test tube, as described in Adleman (1996):

- *Detect.* Given a tube T, say 'yes' if T contains at least one DNA molecule, and say 'no' if it contains none. Tube T is preserved.

The implementation of *detect* is based on PCR. A *program*[5] is a sequence of operations on labelled test tubes. Each statement is of the form:

$$\langle +(T_a, i) \rightarrow T_b; -(T_a, i) \rightarrow T_c; \rangle,$$

where the arrow means "is to be merged with". In other words, one separation and two merges occur for every statement (but note that T_b or T_c may be empty prior to the merge). For clarity,

---

[3]We consider only the case where one variable at a time is tested. More sophisticated operations where multiple DNF minterms are tested simultaneously (see Boneh et al. (1996b)) require more lengthy preparation; thus we argue that the single variable case is not unreasonable for measuring complexity.

[4]We do not consider here whether *Detect* could be used to advantage in the middle of a computation. Apparently, it can be (Lipton 1996a).

[5]The class of programs as given here is slightly different from that given in Adleman (1996). In particular, we insist that a labelled test tube is not re-used after its contents have been used (i.e. "destroyed"). The differences are merely a matter of notation, and inconsequential.

programs can be shown diagrammatically (see Figure 2.1). At the beginning, all test tubes are empty except for T_1, which contains all 2^n DNA strands encoding all possible input vectors $\mathbf{x}$. If at the end of the program execution there is a test tube containing exactly those bit strings which satisfy f, then we say say the program has inverted f, or has solved f. The *size* of a program is considered to be the number of statements (here *Separate* operations) in the program. Since programs are considered to be executed sequentially, the size of a program to invert f is often referred to as the time to solve f. The *width* of a program is the maximum number of test tubes co-existing at any given time.

$$f(\mathbf{x}) = \text{``}0 < \sum_i x_i < 4\text{''}$$

Given $T_1 = \{0,1\}^n$.

$$\langle +(T_1, 1) \to T_3; -(T_1, 1) \to T_2; \rangle$$

$$\langle +(T_2, 2) \to T_5; -(T_2, 2) \to T_4; \rangle$$

$$\vdots$$

$$\langle +(T_9, 4) \to T_T; -(T_9, 4) \to T_T; \rangle$$

$$\langle +(T_{10}, 4) \to T_F; -(T_{10}, 4) \to T_T; \rangle$$

Return T_T.
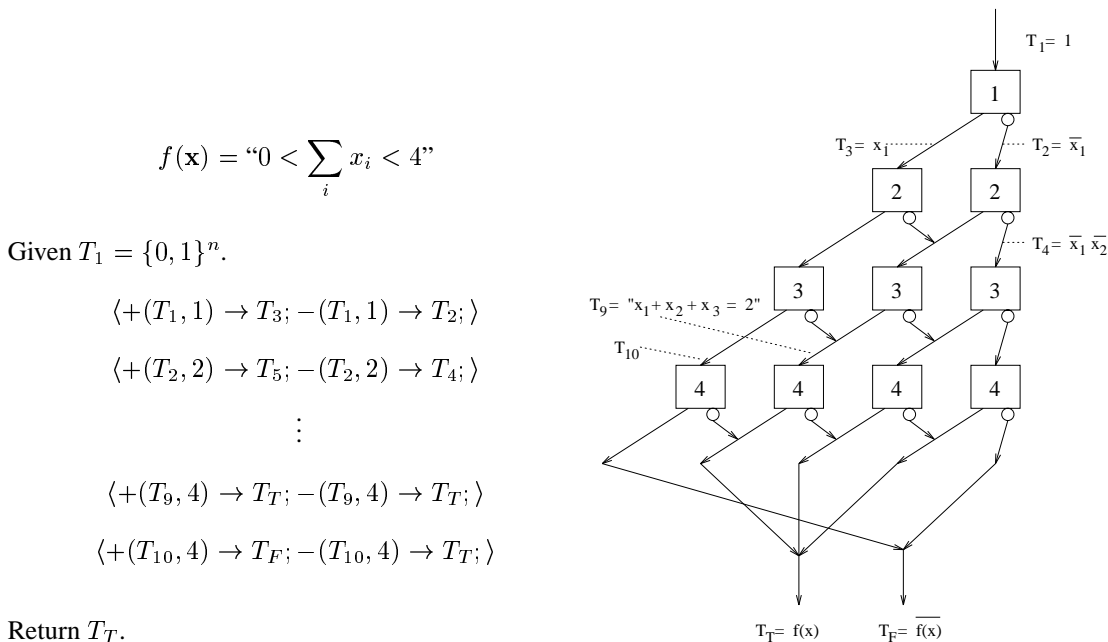
Figure 2.1: Implementing an arbitrary symmetric function in $\frac{n(n+1)}{2}$ separations using the restricted model. Boxes represent separation steps, and arrows represent the test-tubes. Labels, in a few illustrative examples, indicate the logical formula which every strand in the test tube satisfies.

The Unrestricted Model:
The unrestricted model allows one addition type of operation during the computation:

- *Amplify.* Given a tube T produce two tubes T_1 and T_2 with contents identical to T. T is destroyed.

Amplify is implemented using PCR. Programs for the unrestricted model consist of statements similar to those for the restricted model, but with the additional form:

$$\langle T_a \to T_b, T_c; \rangle$$

Here the arrow means, "is to be copied into." Unrestricted model programs can also be shown diagrammatically (see Figure 2.2).

The unrestricted model, unlike the restricted model, can actually "circumvent" the restriction on using only $O(2^n)$ strands, because the number of strands can be doubled with every *amplify*
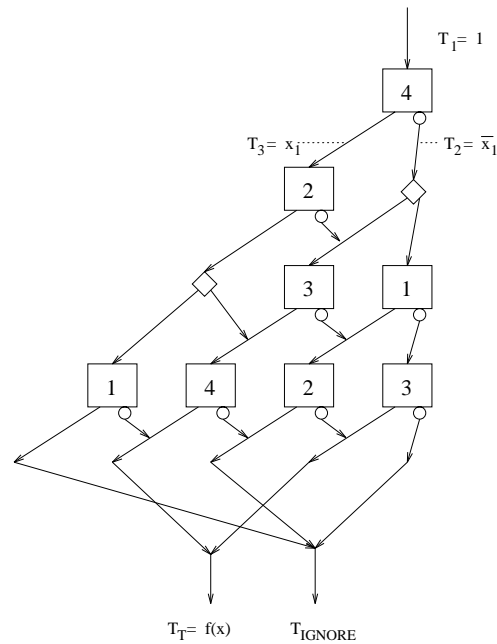
Figure 2.2: Implementing the function $f(x) = x_4(x_2 + x_3) + \overline{x}_4(x_1\overline{x}_2 + \overline{x}_1 x_3 + \overline{x}_3 x_2)$ using the unrestricted model.

operation. We might expect that the unrestricted model is significantly more powerful than the restricted model. Surprisingly, even though we allow the extra volume "for free", there is little benefit.

The Augmented Model:

The augmented model (introduced in Boneh et al. (1996b,a)) does not allow *amplify*, but instead it adds a different type of operation to the restricted model. Here we make use of additional variables $x_{n+1} \ldots$ which are not assigned values by the input.

- *Append[$x_i = v$]*. Given an index $i > n$, a tube T whose strands each encode values for variables $\{x_j\}$ not including x_i, and a value $v \in \{0, 1\}$, modify every strand by ligating the DNA sequence encoding $x_i = v$.

Programs for the augmented model consist of statements similar to those for the restricted model, but with the additional form:

$$\langle T_a \leftarrow \pm i; \rangle$$

Note that *append* cannot assign a value to a variable which has already been set, and similarly we restrict *separate* to cases where on every strand the separation variable has been assigned a value. Only program for which this two properties can be guaranteed are considered valid. Augmented model programs can also be shown diagrammatically (see Figure 2.3).

In the augmented model, like the restricted model, the number of strands remains constant at 2^n. Nevertheless, we will see that the augmented model is more powerful than the unrestricted model, and that the unrestricted model is more powerful than the restricted model.
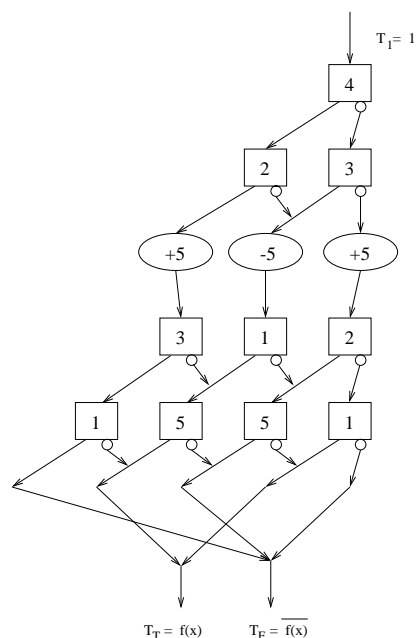
Figure 2.3: An augmented model program implementing a function of unknown importance.

2.2.2   Branching Programs

Since branching programs are not as familiar a model as formulas, finite-state automata, circuits, Turing machines, etc., it is worthwhile to present an exact definition here. We quote from Wegener (1987), p. 414:

> A branching program (BP) is a directed acyclic graph consisting of one source (no predecessor), inner nodes of fan-out 2 labelled by Boolean variables and sinks of fan-out 0 labelled by Boolean constants. The computation starts at the source which is also an inner node. If one reaches an inner node labelled by x_i, one proceeds to the left successor, if the i-th input bit a_i equals 0, and one proceeds to the right successor, if a_i equals 1. The BP computes $f \in B_n$[6] if one reaches for the input a a sink labelled by $f(a)$.

The size of a BP is the number of inner nodes. Many measures of BP have been studied, especially depth and width.

We follow Razborov (1991) in defining a nondeterministic branching program (NBP): we additionally include unlabelled "guessing nodes" of fan-out 2 where both branches are allowed.[7] The NBP computes $f \in B_n$ if by some allowable path one reaches a sink labelled 1 for all $a \in f^{-1}(1)$. The size of an NBP includes the guessing nodes. BP and NBP may be viewed pictorially, as in Figures 2.4 and 2.5, in which the designations "left" and "right" are replaced by "dotted-line" and "solid-line" respectively.

---

[6] B_n is the set of all n-input boolean functions.

[7] This definition of NBP coincides exactly with Meinel's 1-time-only nondeterministic branching programs. His more general definitions seem not to be useful in the context of molecular computing.
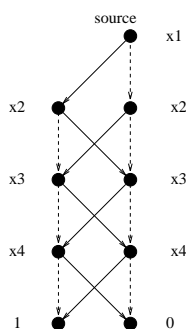
Figure 2.4: Implementing PARITY of 4 variables using a branching program of width 2.
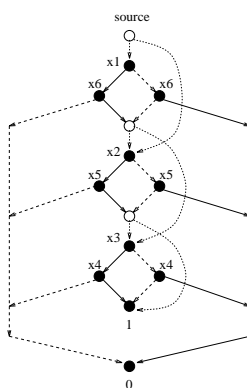
Figure 2.5: Implementing a function using a nondeterministic branching program. $f(x) =$ " $\mathbf{x}$ is palindromic except for isolated (non-adjacent) errors". $NBP(f) \leq 2n + 2$.

We introduce one more modification of branching programs: write-once branching progams (WOBP) are branching programs where the edges may be labelled to assign a value 1 (+) or 0 (-) to any number of gate variables $\{g_i\}$, and where decision nodes may be labelled by a gate variable instead of an input variable *if* all paths to that node assign a unique value to the gate variable. Finally, we also consider circuits where each gate has arbitrary fan-out and computes any boolean function of its 2 inputs.

2.2.3 Correspondence of Models

Restricted Model $\sim$ Branching Programs

In this section we show that the class of functions which the restricted model can invert in a given time are exactly those functions computed by a branching program of the same size.

Examining Figures 2.1 and 2.4, it is clear that not much needs to be proved. The models are essentially identical, except for interpretation. Each separation step corresponds to an inner node of the BP. A strand of DNA corresponds to an input vector for the BP. In summary:

1. If restricted model program P solves f in k steps, then there is a BP G which computes f and is of size k.
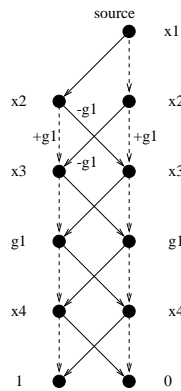
Figure 2.6: A small width-2 WOBP.
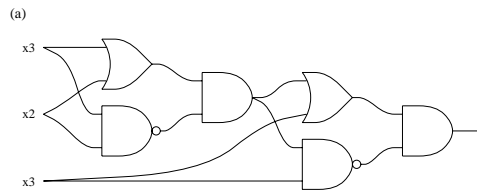
Figure 2.7: A circuit for the XOR of 3 inputs.

2. If BP G computes f and is of size k, then there is a restricted model program P which solves f in k steps.

A single strand of DNA will flow through the test tubes of a restricted model program exactly in the order of inner nodes executed by the associated BP running on an equivalent input vector.[8] Since all possible strands are run in parallel, those that end up in the output test tube T_f are exactly the inputs that the BP accepts; i.e. $f^{-1}(1)$.

Unrestricted Model $\sim$ Nondeterministic Branching Programs

In this section we show that the class of functions which the unrestricted model can invert in a given time are exactly those functions computed by a nondeterministic branching program of the same size.

Examining Figures 2.2 and 2.5, it is clear that not much needs to be proved. We additionally associate *amplify* statements with guessing nodes in the NBP. Just to be clear, we show:

1. If unrestricted model program P solves f in k steps, then there is a NBP G which computes f and is of size k.

2. If NBP G computes f and is of size k, then there is a unrestricted model program P which solves f in k steps.

---

[8]The author is reminded of some friends who needed to transfer a lot of graphics images from San Francisco to Los Angeles. They considered using FTP over the internet, but on second thought realized it would be faster to put the data in their car and drive, so they did. We are doing the same thing here: We physically move a bunch of DNA through the virtual CPU, one gate at a time – but lots of data simultaneously.

We use essentially the same argument as above. However now we say that the set of test tubes which a DNA strand passes through is the same as the set of nodes of the NBP which *could* be activated by the associated input vector. Thus the output test tube contains all strands which *could* cause the NBP to accept; i.e. $f^{-1}(1)$.

Augmented Model $\sim$ Write-Once Branching Programs

In this section we show that the class of functions which the augmented model can invert in a given time are exactly those functions computed by a write-once branching program of the same size.

Examining Figures 2.3 and 2.5, it is clear that not much needs to be proved. We additionally associate *append* statements with writing nodes in the WOBP. Just to be clear, we state:

1. If augmented model program P solves f in k *separation* steps, then there is a WOBP G which computes f and is of size k.

2. If WOBP G computes f and is of size k, then there is a augmented model program P which solves f in k *separation* steps.

We use essentially the same arguments as above; the output test tube contains all strands which cause the WOBP to accept, i.e. $f^{-1}(1)$, and additionally each strand maintains a record all written variables.

The results of Boneh et al. (1996a) can be used to show that WOBPs are as powerful as circuits:

1. If a circuit C or size k solves f, then there is a WOBP G which computes f and is of size $\leq 3k$.

2. If WOBP G computes f and is of size k, then there is a circuit C which solves f and is of size k.

2.2.4 Corollaries and Conclusions

We now have a theoretical handle on precisely what can and cannot be computed by the restricted and unrestricted models. First, by looking at the polynomial size complexity hierarchy, we can separate the classes of functions solvable by the DNA models.

Many useful results follow immediately from the literature on branching programs. Here is a brief sampler:

- poly-size BP are equivalent to log-space non-uniform TM[9] (Meinel 1989).

- poly-size NBP are equivalent to log-space non-uniform NTM (Meinel 1989).

- poly-size circuits[10] are equivalent to poly-time non-uniform TM (Wegener 1987).

- thus poly-size BP $\subseteq$ poly-size NBP $\subseteq$ poly-size circuits, where the inclusions are believed to be proper.

- poly-size, constant-width BP are equivalent to log-depth circuits (Barrington 1986; Cai and Lipton 1989).

---

[9](N)TM = (nondeterministic) Turing machine.

[10]In this note we consider circuits where gates are fan-in 2, arbitrary fan-out, and have arbitrary logic.

function f_n	PARITY	DISTINCT	MAJORITY	SYMMETRIC
$L_{AON}(f)$	n^2	$O(n^2 \log n)$	$O(n^{3.37})$	$O(n^{4.37})$
	n^2	$\Omega(\frac{n^2}{\log n})$	$\Omega(n^2)$	$\Omega(n \log \log n)$
$BP(f)$	$2n-1$		$O(n \log^3 n)$	$O(\frac{n^2}{\log n})$
	$2n-1$	$\Omega(\frac{n^2}{\log^2 n})$	$\Omega(\frac{n \log n}{\log \log n})$	$\Omega(\frac{n \log n}{\log \log n})$
$NBP(f)$	$2n-1$			$O(n^{3/2})$
		$\Omega(\frac{n^{3/2}}{\log n})$	$\Omega(n \log \log \log^* n)$	
$C(f)$	$n-1$	$O(n \log n)$	$O(n)$	$O(n)$
	$n-1$	$\Omega(n)$	$\Omega(n)$	$\Omega(n)$

Table 2.1: Lower and upper bounds on complexities under known models for various functions.

- $\sqrt[3]{C(f)} \preceq NBP(f) \preceq BP(f) \preceq L(f)$ (Razborov 1991)[11].

- $\frac{C(f)}{3} \leq BP(f) \leq L(f) + 1$ (Wegener 1987)[12].

With each of these results there is typically an efficient simulation (Pudlák 1987). Other known linear simulations by branching programs include finite-state automata (FSA) and 2-way finite-state automata (Barrington 1986).

As mentioned earlier, results on polynomial equivalence are only of theoretical and not practical relevance. We would like more exact bounds on the complexity of implementing specific functions. The literature on branching programs gives us some such bounds, although admittedly the knowledge is very incomplete. Some known bounds[13] for a few functions[14] are summarized in Table 2.1.

2.2.5 Discussion

Do we gain anything by using the *amplify* operation? Theoretically, yes, but very little. Contrary to the suggestion in Lipton (1996b), the unrestricted model does not allow us to invert functions defined by circuits in linear time[15]. Furthermore, in addition to concerns about the reliability of

---

[11]C(f) is circuit size, L(f) is AON formula size, etc. $F \preceq G$ means $F = O(G)$.

[12]Note this construction for formulas is better than that given in Lipton (1996b).

[13]See especially Wegener (1987): pp. 76, 85, 143, 243, 247, 261, 440; Razborov (1991): pp. 50, 51; Boppana and Sipser (1990): pp. 793-797. Note Razborov incorrectly quotes the BP lower bound on MAJORITY (Babai et al. 1990). The upper bound comes from Sinha and Thathachar (1994). The upper bound on formulas for symmetric functions follows directly from the upper bound Wegener gives for MAJORITY. The upper bound on circuits for DISTINCT comes from a simple application of SORT, followed by adjacent comparisons; a better bound may be achievable. The upper bound on NBP for symmetric functions uses a construction by Lupanov for switching-and-rectifier circuits (see Razborov (1991)); the construction also works for NBP.

[14]Let $|\mathbf{x}|$ denote the length of $\mathbf{x}$ and $\#\mathbf{x}$ denote the number of 1's in $\mathbf{x}$. Let $m = \frac{n}{2 \log_2 n}$, $|\mathbf{x}_i| = 2 \log_2 n$ and DISTINCT$(\mathbf{x}_1, \ldots, \mathbf{x}_m) = 0$ iff $\exists i \neq j$ s.t. $\mathbf{x}_i = \mathbf{x}_j$. MAJORITY$(\mathbf{x}) = 1$ iff $\#\mathbf{x} \geq \frac{n}{2}$ where $n = |\mathbf{x}|$. PARITY$(\mathbf{x}) = 1$ iff $\#\mathbf{x} \equiv 1 \mod 2$. f is SYMMETRIC if f depends only on $\#\mathbf{x}$. The lower bounds are for almost all symmetric f.

[15]It appears that Lipton realized this shortly after distributing his draft. He later characterizes his constructions in terms of *contact networks*, which are related to branching programs (Lipton 1995).

PCR, we should realize that each *amplify* at least doubles the volume of DNA that we have to handle. After just a few such operations, we could practically be unable to continue the computation. For example, if we conclude for practical reasons that 2^{50} molecules of DNA are the most we can handle in one test tube, then we must be very careful not to exceed this limit when merging the products of amplification[16]. The augmented model of Boneh et al. (1996a), however, both avoids the difficulties of the *amplify* operation and achieves inversion of functions defined by circuits. Another model which achieves inversion of functions defined by circuits is the memory model of Adleman (1996), which can be implemented via site-directed mutagenesis using the methods of Beaver (1996) (who went further to show a full Turing machine simulation).

Because circuits are such a concise representation for most functions of interest, the augmented model seems to provide an effective way to exploit the parallelism of DNA reactions to solve inverse problems. However, for functions represented by circuits of size 1000, the required 3000 laboratory steps is still a lot to ask, especially since each affinity separation and ligation step would take at least an hour if performed by a competent technician according to standard protocols. It is not yet clear what the best biotechnology is for the *separation* and *append* operations, nor what their intrinsic error rates must be. Methods to improve error rates due to misclassification during separations (Karp et al. 1996; Roweis et al. in press) require multiplicative increases in the number of steps, because each separation is repeated enough times to make classification errors rare.

2.3 $0(1)$ **Methods for DNA Computation**

Abstract[17] This section introduces a more novel brand of DNA-based computing wherein the problem to be solved is encoded entirely in the DNA sequences used, and a fixed sequence of experiments is performed. We focus on the experimental technique of *whiplash PCR*, as introduced in Hagiya et al. (in press) for DNA computation, in combination with combinatorial *assembly PCR* to generate structured libraries. We introduce a model of computation based on this technique based on *GOTO graphs*, in which a number of NP-complete problems can be solved in $O(1)$ biosteps, including branching program satisfiability, the independent set problem, and the Hamiltonian path problem. In addition, we propose a simple extension of the experimental technique that allows single DNA strands to simulate the execution of a feedforward circuit, giving rise to a solution to the circuit satisfiability problem in $O(1)$ biosteps.

In an ingenious paper, Hagiya et al. (in press) introduce an experimental technique they call *polymerization stop* and theoretically show how by thermal cycling, individual DNA molecules can compute the output of Boolean μ-formulas (and-or-not formulas in which every variable is

---

[16]On a similar note, even the restricted model can solve f computed by Meinel's more general NBP model, simply by using 2^m times more DNA volume when there are m non-deterministic variables. This allows computation as efficient as circuits, but at the cost of ridiculous amounts of DNA.

[17]Results in this section also appear in Winfree (in press b). Thanks to Masanori Arita, Daisuke Kiga, Kensaku Sakamoto, Shigeyuki Yokoyama, and Masami Hagiya for discussions of their work; and to Len Adleman for suggesting the HPP example and the name "whiplash PCR."

referenced at most once). Because each DNA molecule repetitively forms hairpins so that it can serve simultaneously as both "primer" and "template" for a stopped polymerase reaction, Adleman has dubbed this experimental technique *whiplash PCR*. Hagiya et al. (in press) describe how whiplash PCR can be used to solve the problem of learning μ-formulas given positive and negative data, and more recently Sakamoto et al. (in press) has shown how other NP-complete problems can be solved with whiplash PCR[18].

The motivation for whiplash PCR begins with the interpretation of DNA polymerase as an enzymatic Turing Machine implementing the simply COPY operation. Bennett (1982) goes farther and imagines designing a set of enzymes to simulate the operation of an arbitrary Turing Machine, but these ideas were never implemented because of the difficulty of designing enzymes *de novo*. But is the existing polymerase enzyme's computational capability limited to just copying? Recently, Leete et al. (in press) realized that the hybridization of primers in the polymerase chain reaction (PCR) provides information-based control over the COPY operation, and that complex computations (such as the symbolic expansion of determinants) can be carried out in DNA using a series of PCR reactions. However, this is a very labor-intensive series of laboratory procedures, and it has not yet been attempted experimentally. Hagiya et al. (in press) adds two key insights: (1) that polymerase copying activity (which was initiated by the primer sequence) can be conveniently terminated by a "stop sequence" in the template DNA; and (2) that if the $3'$ end of a DNA strand serves as the same strand's primer, then an individual DNA molecule can be a self-contained computational unit. It was shown how in a single reaction, each DNA strand can independently compute the result of a μ-formula, and how the problem of learning μ-formulas from N positive and negative examples can be solved in in $O(N)$ biosteps. (We use the term "biostep" to refer to a single laboratory procedure. Many chemical reaction steps can take place during a single biostep; in whiplash PCR, the many chemical reactions are sequenced by thermal cycling.)

The DNA used in whiplash PCR has the form $5'$-**stop**$_1$-**new**$_1$-**old**$_1$- $\cdots$ -**stop**$_n$-**new**$_n$-**old**$_n$-**head**-$3'$. When the $3'$ end (head) of the DNA strand anneals to a DNA sequence **old**$_i$, polymerase copies the sequence **new**$_i$, and the polymerase is stopped and dissociates upon encountering the sequence **stop** (for example, because the stop sequence is GGG and the polymerase buffer contains only $A, T,$ and G). The head of the DNA now contains a new sequence. Upon the next thermal cycle, the head can anneal to a different **old** location, and copy the corresponding **new** sequence. We will refer to the basic DNA unit $5'$-**stop**-**new**-**old**-$3'$ as a *frame* and use the notation (**new old**). In general, **boldface** will be used when referring to DNA sequences, while *italics* will be used when referring to logical variables.

We describe by example the method given in Hagiya et al. (in press) by which a single DNA strand computes a μ-formulas during whiplash PCR. Consider the μ-formula $f = (x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_4)$. This can be translated to the decision process shown in Figure 2.8, wherein variable x_1 is checked first; if it is false (written False, 0, or $-$) then variable x_3 is checked, etc. Decision processes of this form are known as *branching programs*[19]; they have already arisen in the study of DNA computing based on affinity separation (Winfree 1996a). Here we have the restriction that each variable be accessed at most once; we call these μ-branching programs. μ-branching programs can represent more functions than μ-formulas; in the absence of this restriction, branching programs are provably more concise than formulas[20].

---

[18]Sakamoto et al. (in press) use the term *successive localized polymerization* to allow for the possibility of intermolecular reactions as well as intramolecular reactions.

[19]Also known as *binary decision diagrams*.

[20]For example, the best known procedure for finding and-or-not formulas implementing symmetric functions results in formulas of size $O(n^{4.37})$, whereas branching programs of size $O(\frac{n^2}{\log n})$ can be achieved.
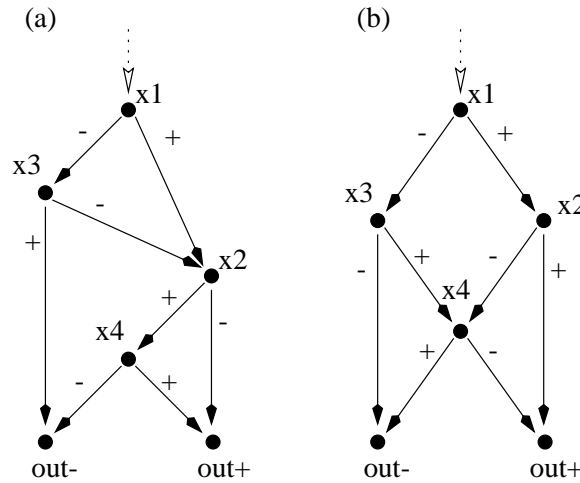
Figure 2.8: (a) A branching program for computing the μ-formula $(x_1 \vee \overline{x_3}) \wedge (\overline{x_2} \vee x_4)$. A possible input would be $x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$, which leads to output $^+$. The computation follows a path through the diagram, and thus can only access variables in the order prescribed. (b) A branching program which does not correspond to a μ-formula.

The translation of an n-variable μ-branching program into DNA makes use of the $3n + 2$ DNA sequences $\{\mathbf{x}_1, \mathbf{x}_1^-, \mathbf{x}_1^+, \cdots, \mathbf{x}_4^+, \mathbf{out}^-, \mathbf{out}^+\}$. Each edge in the diagram, say the $-$ edge from node i to node j, is then converted into a DNA frame $(\mathbf{x}_j \ \mathbf{x}_i^-)$, which may be read as "if x_i is False, check x_j next." A recursive formula is given in Hagiya et al. (in press) that converts any μ-formula directly into a sequence of DNA frames, the *program frames*. To tell the DNA the values of the input variables, we use additional frames of the form $(\mathbf{x}_i^+ \ \mathbf{x}_i)$, read as "$x_i$ has the value True;" these are the *data frames*. The data frames and the program frames are concatenated into a single strand of DNA, with an initial $3'$ head sequence complementary to $\mathbf{x}_1$. Figure 2.9 gives a full set of frames used to implement f and shows how the computation proceeds during whiplash PCR: the head initially anneals to the data region to read the value of x_1; in the next thermal cycle, the head anneals to the frame representing the appropriate edge out of node 1 in the program region, to determine which variable must be checked next; in the next cycle, the head anneals again to the data region, and so on[21]. Because the head might anneal to its previous location (in which case the polymerase is immediately dislodged by the **stop** sequence and nothing happens), the computation proceeds at approximately 1 logical step per two thermocycles. In this fashion, every DNA strand computes in parallel, each containing its own data and its own program.

In the inductive inference problem discussed in Hagiya et al. (in press), one starts with a combinatorial library of DNA representing all μ-formulas of a given size. In each iteration, a positive or negative input example is evaluated by each DNA strand: DNA representing the input is ligated to all remaining DNA strands, which are then evaluated in parallel using whiplash PCR. Those DNA strands computing the correct output value are retained, and the program region is cut from the data and head regions in preparation for the next round of the iteration. After all input

---

[21]The restriction that each variable be used at most once arises because the value of the variable itself, encoded in DNA as $\mathbf{x}_i^{\pm}$, is used to keep track of where the computation is in the decision diagram; if there were two nodes which check variable i, then the computation could return to the wrong place in the diagram because there would be two frames matching $\mathbf{x}_i^{\pm}$.
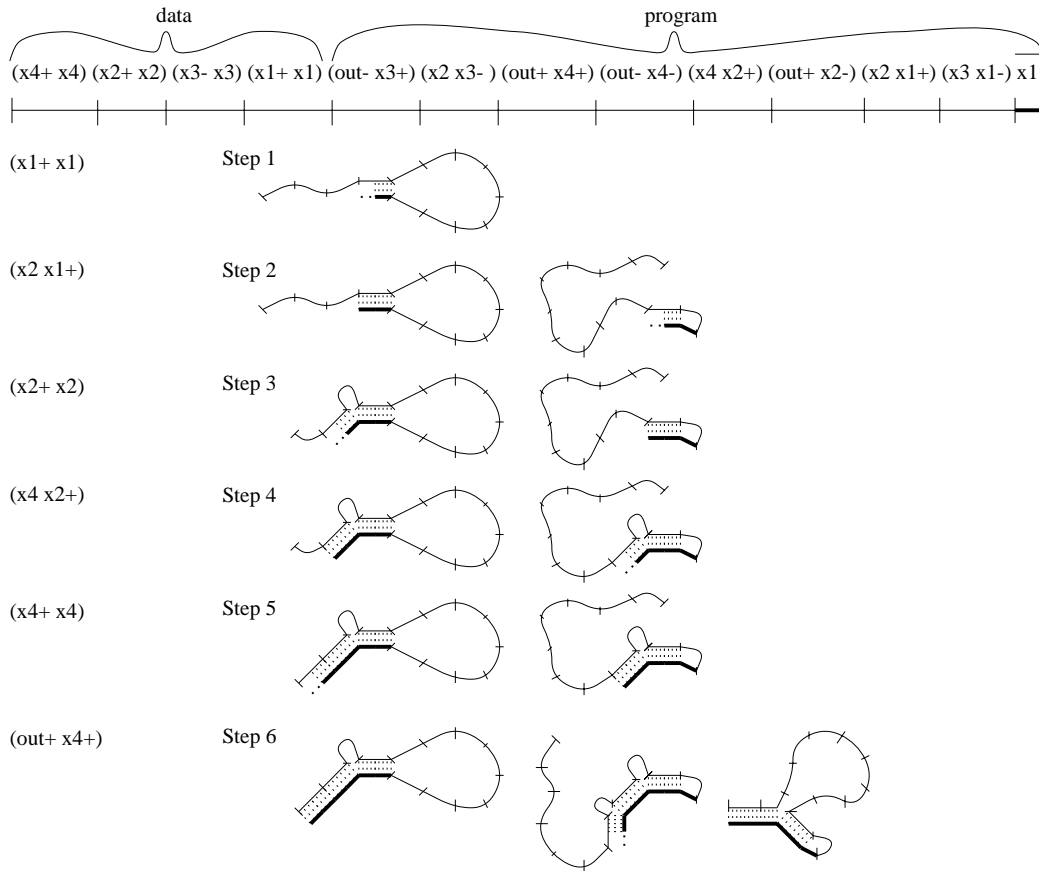
Figure 2.9: Probable secondary structures during the computation of the μ-formula $\left(x_1 \vee \overline{x_3}\right) \wedge \left(\overline{x_2} \vee x_4\right)$ on the input 1101. "Probable" is in the mind of the artist. Note that the tick marks denote the **stop** sequence; because the $3'$ head sequence will never contain the complement to the **stop** sequence, this will be the site of a small bulge in regions that are shown as double-stranded.

examples have been processed, the only DNA programs that remain represent μ-formulas which agree with all examples, and the inductive inference problem has been solved in $O(N)$ biosteps.

By starting with a combinatorial library of DNA representing possible inputs, Sakamoto et al. (in press) describe how whiplash PCR can also be used to solve other NP-complete problems, including conjunctive-normal-form satisfiability (CNF-SAT), Vertex Cover, Direct Sum Cover, and Hamiltonian Path. In the next two sections, we develop similar results for general formula satisfiability (FSAT), branching program satisfiability (BP-SAT), Independent Set, and Hamiltonian Path. We suggest the *assembly graph* formalism for the assembly PCR technique, and the *GOTO graph* formalism for describing computations possible by performing assembly PCR and whiplash PCR followed by a single affinity separation.

2.3.1 Solving FSAT in $O(1)$ biosteps

Even though a single strand of DNA can only compute the result of a μ-formula, it is possible to solve the formula satisfiability problem in $O(1)$ biosteps – without the restriction that each variable can occur at most once.

Consider the Boolean formula $f = (x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$. It is a function of $n = 3$ variables, and it accesses one of them more than once; thus it is not a μ-formula. However, if we introduce the new variables $x_{11} = x_{12} = x_1$, then the same function is computed by the μ-formula $\hat{f} = (x_{11} \vee \overline{x_2}) \wedge (\overline{x_{12}} \vee x_3)$, with the additional constraint that $x_{11} = x_{12}$.

In general, if f is a Boolean formula in n variables in which variable i is accessed q times, then we can construct a μ-formula $\hat{f}$ in $\hat{n} = \sum_{i=1}^{n} \sigma_i$ variables, which computes the identical function for input which is appropriately constrained. Specifically, for each $1 \leq i \leq n$, we require $x_{i1} = \ldots = x_{i\sigma_i}$.

We can use the biochemistry of whiplash PCR to compute the μ-formula, and use the biochemistry of hybridization to generate a combinatorial library of DNA representing all possible inputs which obey the equality constraints. Following Adleman (1994), the combinatorial library consists of DNA representing paths through a graph. We use bipartite *assembly graphs*, in which nodes are either black or white and are labelled by distinct single symbols, and directed edges are labelled by symbol strings (possibly length zero) whose symbols are disjoint from those used at nodes. Each symbol represents a unique sequence of DNA. An oligo is generated for each edge in the graph, using the sequences for the symbols of the origin node, the edge, and the destination node: since the graph is bipartite, edges are either from white nodes to black nodes (in which case "sense" oligos are synthesized), or from black nodes to white nodes (in which case the Watson-Crick complementary "anti-sense" oligos are synthesized). These oligos may be mixed in a single test tube and full-length product may be generated using *assembly PCR*[22] (Stemmer et al. 1995). This reaction creates long "repetitive" DNA, which may then be cut at a restriction site to yield defined-length product, and then made single-stranded. For each path through the graph, the sequence of node and edge symbols on that path will be generated in DNA by assembly PCR; the complementary DNA will also be generated[23]. Figure 2.10 gives an assembly graph for generating all DNA representing inputs where $x_{11} = x_{12}$.
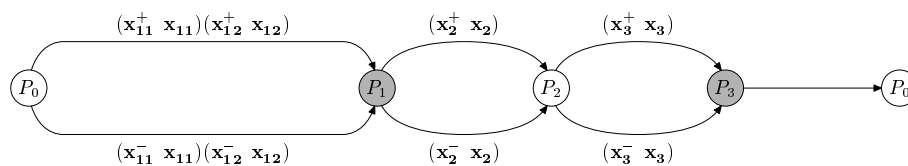
Figure 2.10: An assembly graph for generating input to the formula $(x_1 \vee \overline{x_2}) \wedge (\overline{x_1} \vee x_3)$. Up to $2n + 1$ oligos are required, and additional symbols P_i are used. For convenience, the node P_0 is written twice. Since there will be a restriction site in $\mathbf{P}_0$, this results effectively in paths from the leftmost node to the rightmost.

Thus, for any μ-formula $\hat{f}$, we can generate a combinatorial library of DNA representing all

---

[22]This technique is preferred over annealing and ligation due to its improved yield and accuracy; it was used in Ouyang et al. (1997) to create a full library of 6-bit inputs. Note that if the oligos are simply annealed, there are gaps in the double-stranded DNA; these gaps are filled in by the polymerase during assembly PCR. If, as in Adleman (1994), ligation rather than assembly PCR is preferred, then additional oligos must be generated complementary to the frames on the "anti-sense" strands. Of course, for either ligation or assembly PCR to be effective, careful design of the oligos is required; see, for example Deaton et al. (in press).

[23]To be assembled by ligation, no gaps may be present in the the "sense" strand; therefore all "anti-sense" edges must be labelled by the empty string, or additional oligos complementary to the single-stranded "anti-sense" regions must be synthesized. A general assembly graph can be easily transformed into one suitable for ligation by either of these two modifications.

possible inputs satisfying the equality constraints $\{x_{i1} = \ldots = x_{i\sigma_i}\}$. After assembly of the input DNA, DNA representing $\hat{f}$ can be ligated to the end of all input DNA, the whiplash PCR reaction performed, and DNA whose $3'$ end is $\mathbf{out}^+$ extracted. This DNA contains the input which satisfies the original formula f. We have solved FSAT in $O(1)$ biosteps (granting that the number of thermocycles necessarily will scale with the size of the formula). The exact procedure described above can also be used for the slightly more difficult BP-SAT problem.

2.3.2   Combinatorial Sets of GOTO Programs

We would now like to generalize the techniques used to solve FSAT. To solve FSAT, a sequence of three laboratory procedures was employed: combinatorial generation of DNA by assembly PCR, evaluation of μ-formulas by whiplash PCR, and selection of DNA evaluating to True by affinity separation. Here we introduce a new formalism to describe the computations which can be performed in this manner; this formalism suggests several optimizations and new applications of whiplash PCR.

Our interest comes from the following simple observation: On a given strand of properly constructed DNA, whiplash PCR can be considered as executing a BASIC program consisting entirely of GOTO statements: e.g. the DNA frame $(\mathbf{x}_j\ \mathbf{x}_i)$ can be thought of as "Line i: GOTO line j", or just $i \to j$. The special "line numbers" are $START = 1$, $ACCEPT = \mathbf{out}^+$ and $REJECT = \mathbf{out}^-$. The sequential order in which the GOTO statements appears does not matter, but no line number may appear on the left hand side twice. By using combinatorial synthesis to create a huge number of different programs, and extracting the accepting ones, we are able to solve some interesting mathematical problems. We define a combinatorial set of GOTO programs using a bipartite assembly graph where edges are labelled (possibly with repetition) by GOTO statements and nodes are labelled (uniquely) from P_i. We will insist that all paths generate valid GOTO programs, in which no line number appears twice on the left hand side[24]. This implies, among other things, that the graph has no cycles.

Thus, we consider the following question: Given a graph as defined above, is there a path that generates a GOTO program that reaches $ACCEPT$ when started at line 1? Call this the *GOTO graph satisfaction problem*, or GG-SAT. GG-SAT thus formalizes what can be computed in $O(1)$ biosteps by applying assembly PCR followed by whiplash PCR and affinity separation.

As an example, we will reduce BP-SAT to GG-SAT. Three resource measures of importance are the number of paths through the graph (corresponding to the number of DNA strands generated); the maximal length of the GOTO programs thus generated (corresponding to the length of the DNA strands); and the size, in number of edges, of the GOTO graph (corresponding to the number of DNA oligos that must be synthesized). Then, as shown in Figure 2.11(a), n-variable m-node BP-SAT can be solved by creating 2^n programs of length $2m + n$, using a GOTO graph of size $2(n+m)$. m lines of the program are fixed; the other m lines are generated in independent blocks of σ_i lines, with two possibilities for each.

This notation makes it obvious that the fixed portion of a GOTO graph is redundant; we can reduce each graph to a smaller one by following all the GOTOs in the fixed portion. The example in Figure 2.11(a) reduces to just 3 nodes as shown in Figure 2.11(b). Thus we get the improved theorem that n-variable m-node BP-SAT can be solved by creating 2^n programs of length m using a GOTO graph of size $2n$. The m lines are generated in independent blocks of q lines, with two possibilities for each. Because this decreases both the length of the DNA and the number

---

[24]DNA programs in which a line number appears more than once on the left hand side would execute *probabilistically*.

(a)

input region

program region

$1 \to 6$ $2 \to 8, 3 \to 10$ $4 \to 12, 5 \to 14$

$6 \to 2$ $7 \to 3$ $8 \to 5$ $9 \to 4$ $10 \to 4$ $11 \to 5$ $12 \to -$ $13 \to +$ $14 \to +$ $15 \to -$

$1 \to 7$ $2 \to 9, 3 \to 11$ $4 \to 13, 5 \to 15$

(b)

combined input and program region

$1 \to 2$ $2 \to 5, 3 \to 4$ $4 \to -, 5 \to +$

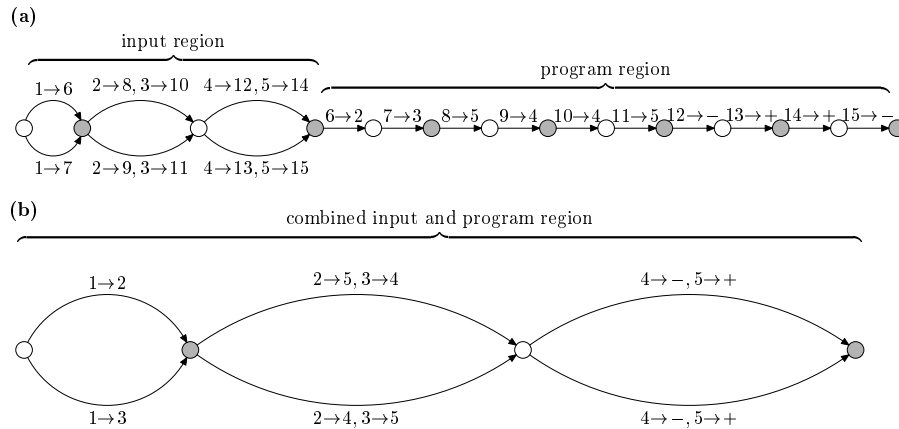$1 \to 3$ $2 \to 4, 3 \to 5$ $4 \to -, 5 \to +$

Figure 2.11: Reducing BP-SAT to GG-SAT: the $n = 3, \hat{n} = 5$ example. (a) The direct construction, combining the assembly graph from Figure 2.10 and the μ-formula program for $\left(x_{11} \vee \overline{x_2}\right) \wedge \left(\overline{x_{12}} \vee x_3\right)$. (b) The optimized construction obtained by following GOTO statements in the fixed region of (a). All GOTO programs are of length 5.

of cycles to complete the program, this construction could be important for experiments solving BP-SAT. It would be interesting to find general polynomial-time algorithms for "optimizing" or "compressing" arbitrary GOTO graphs, in the sense that the new graph solves the same problem but contains fewer paths and/or shorter programs.
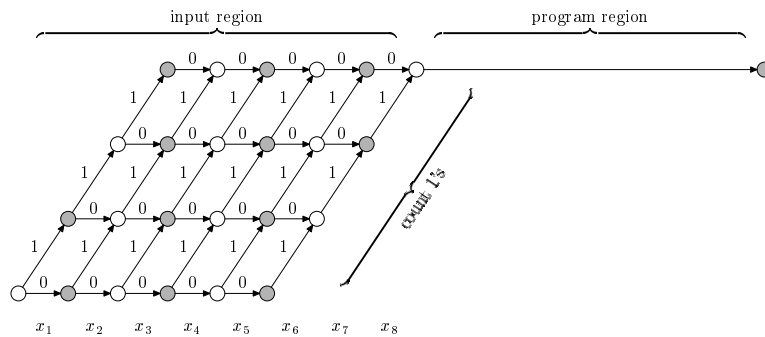
Figure 2.12: A GOTO graph for solving the Independent Set Problem. Inputs are generated in which exactly $k = 3$ out of $n = 8$ variables have value 1. The edge labels "0" and "1" in column i are shorthand for GOTO statements setting the value of variable x_i; as in FSAT, variables which are referenced more than once in the formula must be duplicated, and the corresponding edges in the graph will be labelled with more than one GOTO statement. Note that concentration ratios of the oligos could be adjusted to make all paths equally likely (for ligation-based assembly, at least; it is not so clear for assembly PCR).

However, we are still failing to fully exploit the expressive power of the graph; so far we have considered only essentially linear graphs. In the context of circuit satisfiability, Boneh et al. (1996a) commented that providing a regular language as input to the circuit, rather than

just $\{0,1\}^*$, could for some problems both reduce the size of the circuit and decrease the volume of DNA needed to solve the problem, and that the desired n-bit input can be provided by assembling DNA paths through a graph of size nM, where M is the size of a finite state machine recognizing the regular language. The same comment holds true for BP-SAT. A simple example follows from the ideas in Bach et al. (1996): the polynomial time 2SAT problem becomes NP-complete when given the restriction that satisfying solutions must have exactly k ones. An instance is the Independent Set Problem, which asks, given an undirected graph and an integer k, is there a subset of k vertices which have no edges among themselves? The 2-CNF formula we will use for this problem is

$$\wedge_{s=1}^{e}\left(\overline{x_{i_s}} \vee \overline{x_{j_s}}\right)$$

where the graph has edges $[i_1, j_1] \ldots [i_e, j_e]$ and x_i indicates membership in the independent set. The formula simply checks that no two chosen vertices have an edge between them. To solve the problem, we ask for a solution to this formula in which *exactly k* variables are 1. This is done in DNA by generating only inputs with k variables set. A GOTO graph for this problem is shown in Figure 2.12; variables used more than once must be duplicated, and the fixed GOTO statements in the "program region" can be eliminated just as in the BP-SAT optimization.
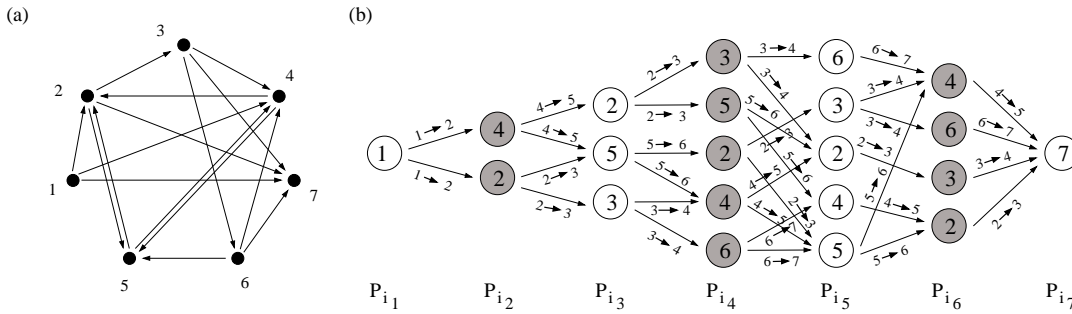
Figure 2.13: Solving the Hamiltonian Path Problem: A graph G (a) and its corresponding GOTO graph GG (b). This is Adleman's example with 2 additional edges added to prevent pruning from simplifying the GOTO graph to triviality. For convenience the nodes show only the vertex index i, and not the full symbol P_{i_k}.

As a final example, we consider the Hamiltonian Path Problem (HPP) solved in Adleman (1994). Our procedure begins by converting (in polynomial time) the original graph G into a GOTO graph GG. Suppose G has n vertices; then GG will have n^2 vertices, arranged in layers, such that if there is an edge $[i, j]$ in G, then in the GOTO graph, for each $k \in \{2 \cdots n\}$ there is an edge $[P_{i_{k-1}}, P_{j_k}]$, labelled $i \to (i+1)$ (with $ACCEPT = n$). Since we are only interested in paths from vertex 1 to vertex n, we prune the new graph to include only vertices which may be reached from P_{1_1} and which may reach P_{n_n}; this dynamic programming problem takes time $O(n^2)$ on an electronic computer. We now have the GOTO graph GG, as shown in Figure 2.13. If G has E edges, then GG requires less than E^2 oligos.

Every path through GG represents a length n path through G from vertex 1 to vertex n. A Hamiltonian path will contain, in some order, the frames

$$\{1 \to 2, 2 \to 3, \cdots, (n-1) \to ACCEPT\},$$

and thus the GOTO program, as executed by whiplash PCR, will proceed to $ACCEPT$. All other paths will duplicate some frame and lack another – these GOTO programs will terminate and never reach $ACCEPT$. Consequently, extraction of DNA containing the $ACCEPT$ sequence will identify the Hamiltonian path, and we have solved HPP in $O(1)$ steps.

2.3.3 Single-Strand Computation of Boolean Circuits

Using whiplash PCR in the manner suggested in Hagiya et al. (in press), where exactly one symbol is copied in each polymerization stop step, gives each strand exactly the computational power of a GOTO program, and no more. However, whiplash PCR may give each strand more computational power, if copying more than one symbol is experimentally feasible. The idea is this: when the head of the DNA strand is being extended, it might not only change the "state" of the head but also add a new "program" frame.

Suppose for the moment that the variables x_i are encoded by $\mathbf{x}_i, \mathbf{x}_i^+, \mathbf{x}_i^-$ using A, T, and C, and that the new *gate* variables g_i are encoded by $\mathbf{g}_i, \mathbf{g}_i^+, \mathbf{g}_i^-$ using exclusively A and T. G and C are respectively used for representing the **stop** sequence and its complement. The polymerization buffer still includes A, T, and G, but not C. The restricted alphabet used for the gate symbols makes designing DNA sequences a more difficult task[25], but it is necessary for the construction we give below because now a gate symbol can be copied by polymerase *twice* during whiplash PCR.

In our original discussion of branching programs, a $+$ edge from the node reading x_7 to the node reading x_4 would be encoded by the frame $(\mathbf{x}_4\ \mathbf{x}_7^+)$. During biochemical execution with whiplash PCR, a transition through this edge would entail hairpin formation with binding to $\mathbf{x}_7^+$ and polymerase extension copying $\mathbf{x}_4$, as shown in Figure 2.14(a). Our new proposal involves copying more than $\mathbf{x}_4$ during the polymerase extension, thereby memorizing an intermediate result of the computation. In Figure 2.14(b) we show the execution of an *enhanced frame* $(\mathbf{x}_4\ \overline{(\mathbf{g}_8^+\ \mathbf{g}_8)}\ \overline{(\mathbf{g}_5^-\ \mathbf{g}_5)}\ \mathbf{x}_7^+)$. Here, the original DNA encodes for the "anti-sense" of a valid frame, and thus the frame is inactive, or *hidden*. The two hidden frames present here are intended to assign values to new variables g_5 and g_8, but that assignment will not become effective while the frame is still hidden. However, if the enhanced frame is executed, the hidden frames are copied as "sense" frames onto the growing $3'$ end of the DNA, thus activating the hidden frames for potential future use. The final $3'$ sequence of the DNA will still be $\mathbf{x}_4$, which will determine the immediate course of the computation as usual.

At subsequent points in the evaluation, reference can be made to look for the values of g_5 or g_8. These values will be found by the head hybridizing to the newly activated frames and copying to the GGG stop sequence – only now the head will not be hybridizing to the "input" part of the DNA, but to part of the growing "head history" itself.

What is the use of activating hidden frames? The possibility of memorizing intermediate results gives rise to a model of computation that we call *write-once branching programs* (WOBP)[26]. Each node still has two outgoing edges, one labeled $+$ and the other $-$; however, edges may now also have the additional labels $\pm g_i$, which indicate that the variable g_i is to be assigned the value

---

[25]An expanded DNA alphabet, making use of artificial base pairs which are both highly specific and can be incorporated by DNA polymerase, would allow greater flexibility in sequence design; indeed, Sakamoto et al. (in press) reports preliminary studies of using iso-C and iso-G (Switzer et al. 1993) in whiplash PCR. If this chemistry is successful, the variables x_i and g_i could be encoded using A, T, C, and G; the **stop** sequence could be iso-G-iso-G-iso-G and its complement iso-C-iso-C-iso-C; and the polymerization buffer could contain A, T, C, G, and iso-G.

[26]This model can also be used to describe DNA computation performed by a sequence of affinity separations and ligations, as in Boneh et al. (1996a).
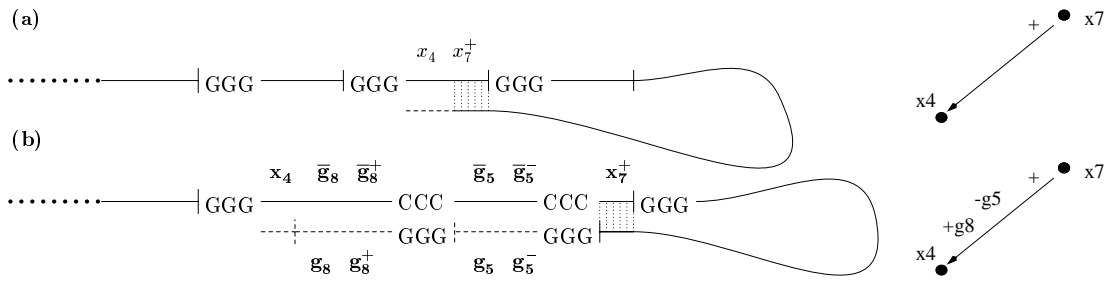
Figure 2.14: (a) The polymerization stop step on a standard frame, where a single symbol is copied, and its representation as an edge in a BP. (b) The polymerization stop step on an enhanced frame, where two hidden frames are made active, and its representation as an edge in a WOBP.

$+$ or $-$. For implementation using whiplash PCR, a restriction is imposed: again, a given variable may be read at most once, and nodes may be labeled to read any input variable x_i or any gate variable g_i, so long as all paths to a given node have assigned exactly one value to the gate variable being read[27]. We call these restricted programs μ-WOBP.
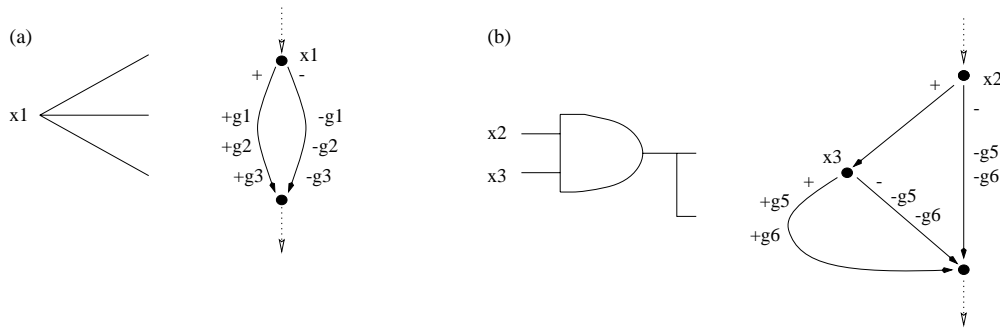
Figure 2.15: (a) Input variables with multiple fan-out are handled by reading them once, and writing multiple distinct gate variables which may subsequently be read once each. (b) The translation of a gate with fan-out 2 into a write-once branching program requires two decision nodes (only one of which is guaranteed to be used). Two new gate variables are written. To translate an entire circuit, first the input variables and then the gates would be processed in linear order in the branching program. Clearly, much more efficient translations are possible; for example, gates with fan-out 1 need not be memorized.

μ-WOBP are at least as concise as circuits[28]; a circuit with n inputs accessed in total $\hat{n}$ times, and g gates with total gate fan-out p can be implemented in a μ-WOBP using no more than $n + 2g$ nodes and $\hat{n} + p$ gate variables[29]. The simple construction uses the building blocks shown in

---

[27]Again, we have a probabilistic model if this restriction is violated.

[28]The converse is also true: a circuit can be constructed in which (usually) two gates are used for each edge in the WOBP to test if the edge was traversed during computation. Thus a circuit with $3m$ gates can be constructed from a WOBP with m nodes.

[29]Just $2g$ nodes and p gate variables are required if we allow preparing the input with duplicated variables, as in the FSAT construction.

Figure 2.15. First, each input variable x_i is read and duplicated into σ_i new variables, so every subsequent read uses a unique variable. Then, each circuit gate is processed in turn, and its output is stored in a new gate variable (or variables, if the gate has fanout greater than unity). The translation of a small circuit is shown in Figure 2.16. Thus, we can theoretically solve the circuit-SAT problem in "one pot" using whiplash PCR.

In the case shown in Figure 2.16, a much smaller μ-WOBP (essentially a BP) exists which computes the same function, pointing out that our construction of a μ-WOBP from a circuit is not the most efficient construction possible. However, for more difficult problems, circuits can be much more efficient than branching programs[30]. This means that a fixed size CSAT problem may be more difficult than a BP-SAT problem of the same size.

One serious concern is that the problem of secondary structure interfering with the progress of the computation is made worse. First, "inopportune" hybridization now involves much longer subsequences, resulting in many thermocycles in which no progress is made. Secondly, newly activated frames are located in the "head history" region of the DNA, which is likely to be involved in secondary structure. Experimental investigation is required to see how serious the problems will be.

2.3.4   Conclusions and Future Directions

Like other forms of DNA computation, it seems that whiplash PCR can't by itself compete with electronic circuits unless there are significant advances in the control of the biochemistry. However, the computational power of whiplash PCR – in theory – suggests that "one-pot" biochemical reactions have more potential for computation than previously thought. Conceivably, whiplash PCR could be combined with other kinds of DNA processing – either stepwise or within the "one pot" biochemical reaction. For example, we can consider modifications of whiplash PCR wherein DNA strands not only grow though polymerization, but also *shrink* due to other enzyme activity (e.g. restriction endonucleases or topoisomerases). An open theoretical question is how to use non-determinism during whiplash PCR: we have already discussed the case where the solution to a problem is found by first using nondeterministic steps in the generation of the DNA, and then using deterministic steps during the execution of the program, but whiplash PCR could equally well be used to perform nondeterministic steps by having multiple frames matching the current head state.

---

[30]As a simple example, an arbitrary symmetric function can be implemented in a circuit of size $O(n)$, but the best construction for branching programs requires $O(\frac{n^2}{\log n})$ nodes.
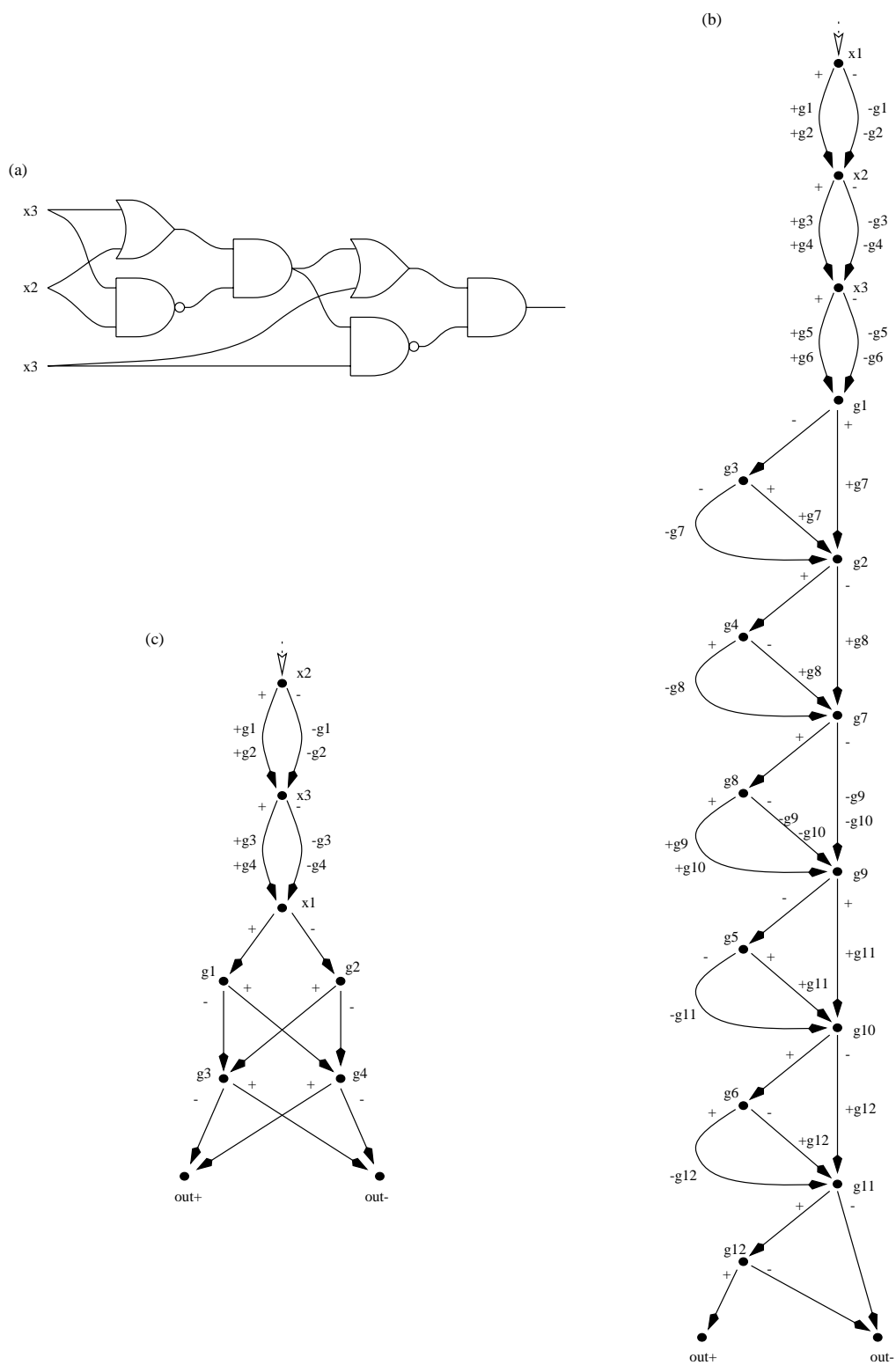
Figure 2.16: The translation of a 3 input, 6 gate XOR circuit into a μ-WOBP. (a) the circuit, (b) the μ-WOBP generated by our construction, (c) a much simpler μ-WOBP generated by hand.

Chapter 3   Models of Computation by Self-Assembly

3.1   2D Self-Assembly for Computation

Abstract[1] This section informally explores the power of annealing and ligation for DNA computation. (The following two sections will explore these notions more formally.) The first step of Adleman's molecular solution to the Hamiltonian Path Problem involves the creation of a combinatorial library of DNA by means of directed self-assembly, followed by ligation. Experimentally, DNA annealing can produce many unusual structures in addition to the usual B-form double helix, so we wonder if they can be used to advantage for computation. We conclude, in fact, that annealing and ligation alone are theoretically capable of universal computation.

When Adleman introduced the paradigm of using DNA to solve combinatorial problems Adleman (1994), his computational scheme involved two distinct stages. To solve the directed Hamiltonian path problem, he first mixed together in a test tube a carefully designed set of DNA oligonucleotide "building blocks", which anneal to each other and are ligated to create long strands of DNA representing paths through the given graph. After this ligation stage, there ensue n steps of affinity purification, whereby exactly the strands representing Hamiltonian paths are separated into a test tube ("the answer").

Lipton (1995) subsequently refined the formalism for DNA-based computation. He did away with Adleman's first stage, ligation, and replaced it by starting all computations with a fixed set of DNA strands representing all n-bit strings. Lipton expanded on Adleman's second stage, separation, where he showed how all solutions to a given boolean formula f can be separated into a test tube ("the answer"). The cost for the generality of this method, even when using the improvements of Boneh et al. (1996a), is indicated by considering solving the Hamiltonian path problem: a straightforward method[2] takes about n^3 separation steps using Lipton's approach, compared to the n steps used by Adleman.

We can conclude from this circumstantial evidence that much of the physical computational power Adleman was exploiting was in his first stage, where annealing and ligation were used. Lipton has explored the power of generalizing Adleman's second stage; we would like now to

---

[1]Results in this section also appear in Winfree (1996b). Thanks to Paul W. K. Rothemund for the discussions at the Red Door Cafe that lead to this work, and to Nadrian C. Seeman for suggesting the use of the double-crossover molecule.

[2]Let the graph have n vertices and e edges; $e \leq n^2$. The best boolean circuit I could devise uses $O(en \log n)$ gates to verify a Hamiltonian path. Another issue is that Adleman's ligation stage requires the synthesis of about $O(n + e)$ oligonucleotides, which is $O(n^2)$ if $e = O(n^2)$; whereas Lipton needs only about $4n \log n$ oligonucleotides to create his standard initial test tube of DNA. However, technology is becoming readily available for synthesizing many oligonucleotides in parallel very quickly (see *e.g.* Chetverin and Kramer (1994)); the same cannot be said for the affinity purification steps, which will likely remain expensive. Comparing volume for a graph with $n/2$ edges out of each vertex, Adleman's method uses volume roughly proportional to $(\frac{n}{2})^n$, while Lipton's method uses a volume of $2^{n \log n}$, since it takes $n \log n$ input variable bits to specify a potential path.

explore the power of generalizing Adleman's first stage.

An immediate stumbling block is that the chemistry of annealing is not fully understood. At best we can try to define some conditions under which the reactions are predictable, or at least under which it is reasonable to expect that the reactions could be made to be predictable.

3.1.1 Some Basic Annealing Reactions

The fundamental chemistry of DNA is based on the double helix and the principle of complementarity. Each strand of DNA is a covalently linked polymer, where each unit consists of a constant part (the sugar-phosphate "backbone") and one of either adenine, thymine, cytosine, or guanine (the bases A, T, C, G). Each strand is oriented; it has a 3' and a 5' end. When DNA forms a double-stranded helix, the strands must be anti-parallel, and complementary bases align (A with T, C with G); such strands are called Watson-Crick complementary sequences. DNA also takes on more complicated configurations, including triple helix, quad helix, super-coiled, and branched.

A surprising number of possibilities are available, some of which one may want, and many of which one may not want. DNA is a particularly easy molecule to work with, because it has evolved to be stable, typically unreactive, yet manipulable. RNA and protein, which have evolved to serve many enzymatic functions, are far more reactive, and thus it is less easy to predict how novel designs will behave in an experiment.

I will now comment on some reactions we may wish to exploit, presented in cartoon fashion (Figure 3.1). I will have to be more detailed with the reactions involved in the main thrust of this paper, where their computation-universality is demonstrated.

(A) This is the canonical annealing reaction for DNA. Two strands with complementary subsequences will form hydrogen bonds and hybridize at the matching base pairs. The rate constants for this reaction, which is reversible, depend on the temperature and salt concentrations, among other things. The melting temperature, above which the complex is not stable, depends upon the number of matching base pairs.

(B) A special case of the above, where the matched region occurs at the ends. Note that the two "sticky ends" (unmatched sequences) are available for further reactions with more DNA.

(C) The above reaction can be used to join two double-stranded DNA molecules with complementary sticky ends. If ligase is present in the solution, the nicks in the backbone of the product will be repaired by the formation of a covalent bond, resulting in two continuous strands.

(D) If mismatches occur flanked by matching regions, the unmatched DNA can bubble out.

(E) As above, except that the mismatch occurs here on both sides. Whether this structure is stable depends critically on the temperature and concentration of salts. For example, a rule of thumb is that the difference in melting temperature between a perfectly matched structure and an imperfectly matched structure is 1 degree per 1% mismatch (Wetmur 1991).

(F) This is the simplest DNA branched junction. The assembly of these structures consists of course of sequential steps; only the end product is shown. This 3-armed junction is probably floppy. However, how floppy it is depends upon the exact sequence of base pairs in the oligonucleotides.

(G) This 4-armed junction is commonly known as a Holliday junction. The two horizontal strands tend not to be parallel, but skew. If the sequences along both strands are homologous, then a phenomenon called branch migration can occur, in which the crossover point drifts right or left.

(H) This is the most complicated structure we will consider. We will put it to good use later. It has been found to be fairly rigid and planar (Fu and Seeman 1993). Note the sticky ends. Other related double-crossover junctions are possible, depending upon the number of half-turns present in the helical regions. Seeman calls this molecule "DAE" for double-crossover, antiparallel helical strands, even number of half-turns between crossovers. "DAO", with an odd number of half-turns between the crossovers, has an interesting topological difference: It consists of only 4 strands.

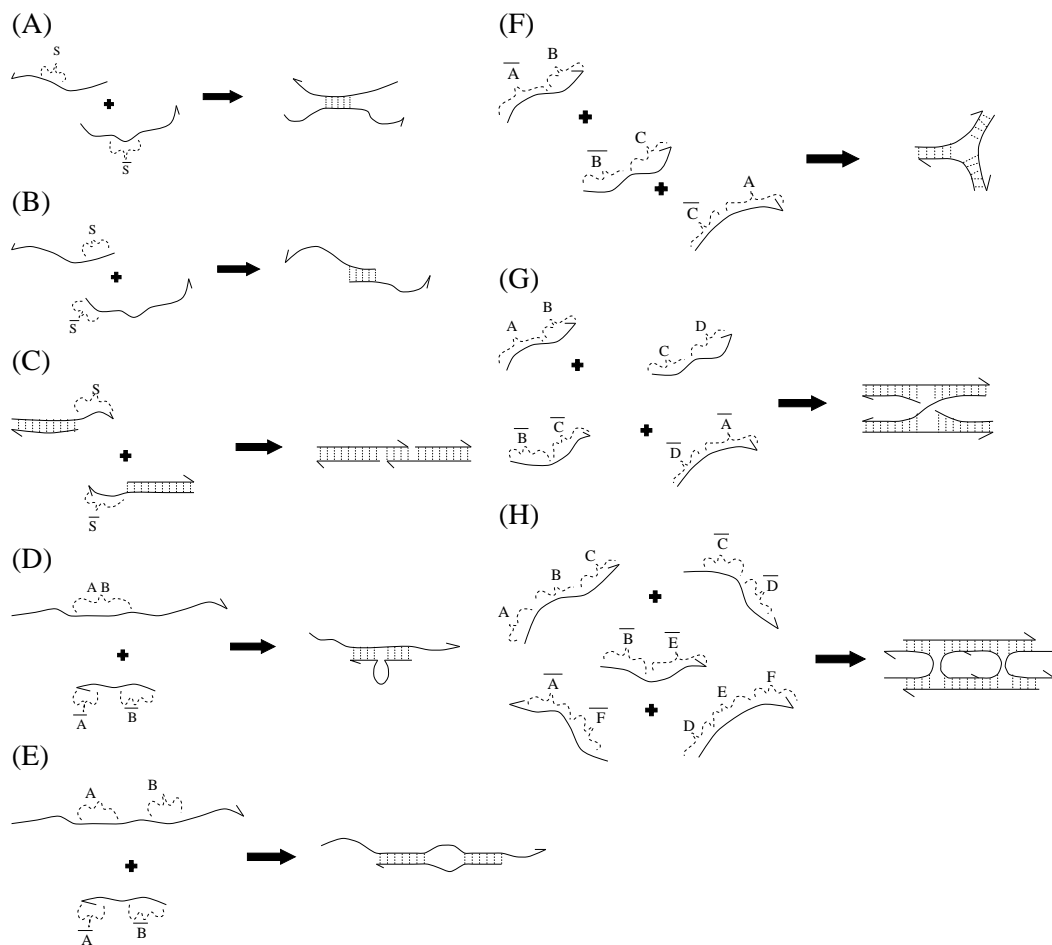Figure 3.1: Some basic types of annealing reaction. Curves represent single strands of DNA oligonucleotide. The half arrow-head represents the 3' end of the strand. Small lines between strands represent hydrogen bonds joining the strands. The helical structure of the DNA is not represented visually. Letters signify sequence motifs. A bar above a letter signifies the Watson-Crick complement of the motif.

All of the structures above have been made in the lab and their structures verified (see, for example, Fu and Seeman (1993)).

We would ultimately like a theory which could tell us, given a set of oligos, a temperature, and salt concentrations, what stable structures will form, as well as the kinetics. But this is a very complex task!

3.1.2 Operations Using Linear DNA

We will first briefly consider what computations can be performed using annealing and ligation of strictly linear DNA molecules. Many of the possibilities have already been discussed by other authors. For example, the techniques used by Adleman (1994) allow for the construction of all DNA representing strings accepted by a finite-state automata (also known as a regular language), using the annealing reactions (B) and (C) above. This is important, because it allows us to create a well-defined, somewhat interesting set of inputs on which to compute in parallel. Beaver (1996) has discussed how, in conjunction with polymerase, reactions (D) and (E) can be used to make copies of DNA with context-sensitive insertion, deletion, and replacement of substrings. In light of these powerful operations, it seems plausible that a "one-pot" linear DNA reaction could be designed which performs universal computation.

3.1.3 Operations Using Branched DNA

There are many possibilities for computation using branched DNA. However, since the general chemistry is not well understood, we will try to avoid ungrounded speculation by focusing on one concrete possibility. The rest of this section[3] will concentrate on how to assemble a large "weave" of branched DNA which simulates the operation of a one-dimensional cellular automaton.

Background: Blocked Cellular Automata

This section develops a formal model of computation called blocked cellular automata (BCA[4]). We will later show how BCA can be simulated by DNA.

The operation of a BCA is diagrammed in Figure 3.2. As in the Turing Machine model, information is stored in an infinite one-dimensional tape, where each cell contains one of a finite set of symbols. The computation proceeds in steps, where in each step the entire tape is translated, according to a given rule table, into a new tape. The translation occurs locally and in parallel; pairs of two cells are read, and which two symbols are written is governed by look-up in a rule table[5]. It is of critical importance that the reading frame (which cells are paired together) strictly alternates from step to step.

The set of entries $\{(x, y) \rightarrow (u, v)\}$ is called the rule table, or the program, of the BCA. By appropriately designing the rule table, the BCA can be made to perform useful computation. In fact, BCA are computationally universal. A BCA with $k + 3s$ symbols can simulate in linear time the operation of a Turing Machine with k tape symbols and s head states – the proof is analogous to that in Lindgren and Nordahl (1990). Thus we can conclude that a BCA can be used to answer any

---

[3] The inspiration for this approach comes from the proof of the undecidability of the Tiling Problem (see Grünbaum and Shephard (1986), Chapter 11).

[4] BCA (Wolfram 1994) are also known as *partitioning* CA (Margolus 1984) and as 2-body CA or particle machines. They generalize the lattice gas model (Hardy et al. 1976), and are commonly studied in two dimensions.

[5] If the table contains multiple entries for a given pair of read symbols, then the BCA is said to be nondeterministic.
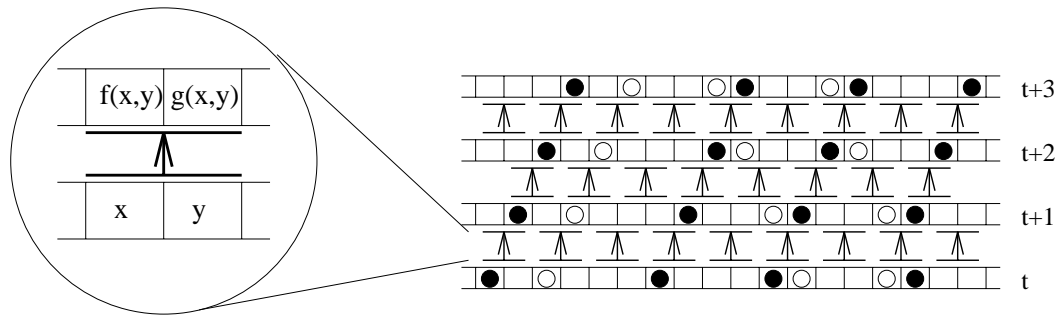
Figure 3.2: Operation of a BCA. The tape of a BCA, divided into cells, is shown at the bottom right. Each cell contains one of three symbols: blank, black dot, or white dot. The tapes at successive time steps are stacked vertically above the initial tape. The inset, left, details the form of a rule table entry, which governs how new tapes are created.

question which can be phrased in terms of a computer program. Small BCA have been designed which sort lists of integers, compute primes, and many other tasks.

A few more comments are in order concerning the abstract model of blocked cellular automata. First we consider the finite-size case. In any attempted implementation of a BCA, we cannot actually construct an infinite tape. Thus boundary conditions become important. We consider the following cases:

(a) No update of boundaries. We start with a finite tape of length $2n$; at each step the tape become 2 cells shorter; and after n steps the computation can proceed no further. This case is not universal.

(b) Inactive boundary conditions. Whenever there is an unpaired cell at either end of the tape, it is copied verbatim onto the new tape. The tape remains always the same size (n cells), and thus there are only k^n possible tapes. As the computation must begin to cycle after k^n steps, this case is also not universal.

(c) Periodic initial conditions. On either side of the input cells we specify a repeating pattern of symbols. Starting with just one copy of the periodic block on either side of the input, computation proceeds as in (a), but if the tape gets too short, we add another copy of the periodic block to either side of the input tape and start the computation anew[6]. This case *is* universal.

(d) Self-regulated boundary conditions. Depending upon what symbol is in the boundary cell, the new tape will either shrink (as in (a)) or expand by appending a new cell to the end of the tape. This case is also universal.

Finally, a word on how an answer is obtained from the BCA. This is a matter of convention. Typically, when the computation is done, the answer is written on the final tape. But how is it known when the computation is done? One possibility is that the tape stops changing; the system has reached a fixed-point. However in this paper we will consider that a computation is done when a special symbol, called the *halting symbol*, has been written for the first time anywhere on the tape.

---

[6]By memorizing boundary cells, we can avoid re-computing any cells and make the computation more efficient.

Simulation of BCA by DNA

We will now show how to use DNA to construct a BCA. In this section we will optimistically show what chemical reactions we *hope* will occur; in the following section we consider potential difficulties in finding conditions such that they will in fact occur as we have described.

The DNA representation of the BCA tape is a little counter-intuitive, so we will explain by example. Figure 3.3 shows part of the DNA molecule encoding the initial tape (the input to the computation). To each tape symbol corresponds a short oligonucleotide sequence, which appears in the initial molecule as a sticky end overhang in the appropriate positions. The rest of the DNA in each segment does not vary with content, and is chosen to maximize structural stability. Note that the reading frame is implicit in the structural form of the DNA. Although Figure 3.3 is schematic, the 2D picture *is* meant to imply that the whole DNA complex is roughly planar. This is critical, and luckily, it is physically plausible.
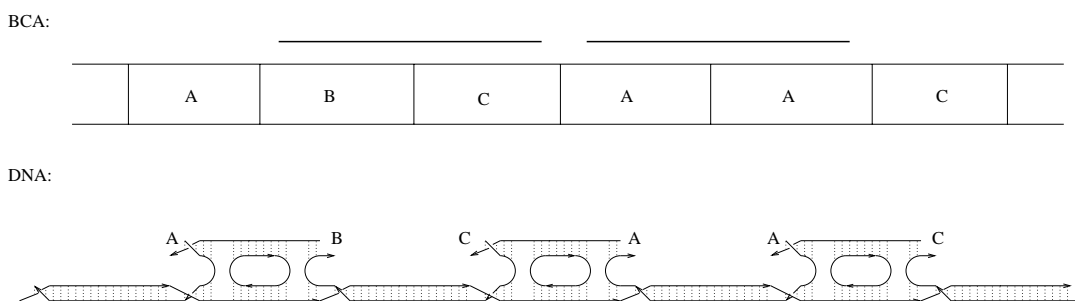

Figure 3.3: Encoding the initial tape in a DNA molecule. The sequence of sticky ends in the initial molecule encodes the initial tape of the BCA. Thus 'A' denotes a symbol in the BCA diagram, whereas in the DNA diagram it denotes the unique sequence of bases associated with that symbol.

There are a variety of ways to make the initial molecule. Note that the initial molecule can be thought of as consisting of several double crossover junctions (from Figure 3.1H, with the modification that the top and bottom strands are made to be an odd number of half-turns in length – see Figure 3.6 for detail) linked together by pieces of linear helical DNA. The sticky ends can be designed such that only this unique molecule will self-assemble[7]. Ligase can be added to make the segments of the initial molecule covalently bonded.

We will now explain how the program, that is the rule table, of the BCA is represented in DNA. For each rule, *e.g.* $(x, y) \rightarrow (u, v)$, we create a double crossover molecule whose sticky ends on one helix are $\overline{x}$ and $\overline{y}$, and on the other helix u and v[8] (see Figure 3.6). All such rule molecules are added to the solution containing the initial molecule. As shown in Figure 3.4, what is required for computation is that rule molecules will anneal into position if and only if both sticky ends match.

Eventually, a triangular lattice of linked DNA will form, simulating a triangular region of a BCA corresponding to boundary conditions (a) or (c) in Section 3.1.3 above (see Figure 3.5). Boundary conditions (b) and (d) can be simulated by using special rule molecules for the edge of the lattice; the details are not presented here. Note that each level of the lattice has a single strand

---

[7]It is easy to see that sticky end sequences can be chosen, using the same techniques as Adleman (see Section 3.1.2), such that a periodic initial molecule will form, creating periodic initial conditions as mentioned in Section 3.1.3 (c) above. Similarly, a regular language of inputs could be made in parallel.

[8]The lengths of all parts of the rule molecules are chosen to be constant for simplicity, but it is conceivable that by using variable length as well as sequence to encode symbols, greater specificity could be achieved.
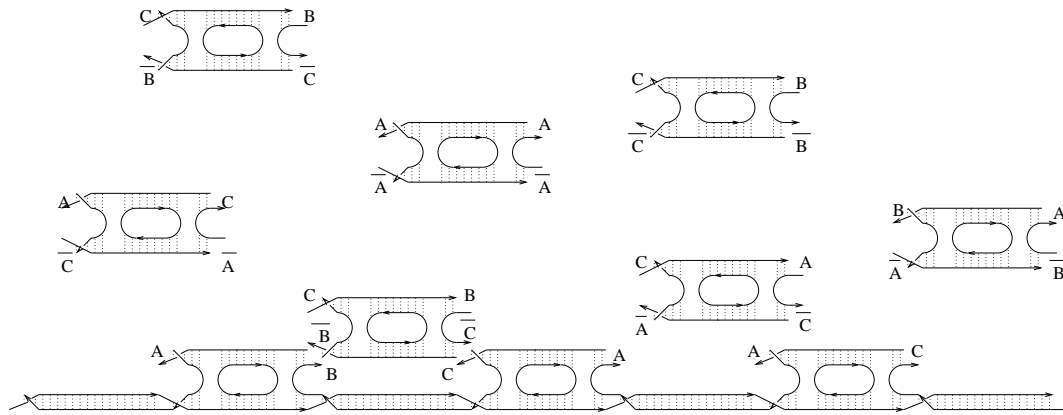
Figure 3.4: Rule table molecules assemble into the lattice. We see free-floating rule table molecules above and the initial molecule at the bottom (both correspond to the BCA in Figure 3.2). A rule table molecule, with sticky ends $\overline{B}$ and $\overline{C}$, is about to anneal to the initial molecule. At the left, a rule molecule which matches only at $\overline{A}$ will ultimately not stick. Note that the rule molecule with sticky ends $\overline{A}$ and $\overline{A}$ will also not stick, because the orientation of its strands is wrong; this rule molecule will be useful on alternate levels of the lattice.

of DNA which travels the entire length of the lattice at that level, and where the coded symbols occur in the sequence in in which they occur in the BCA at time t.
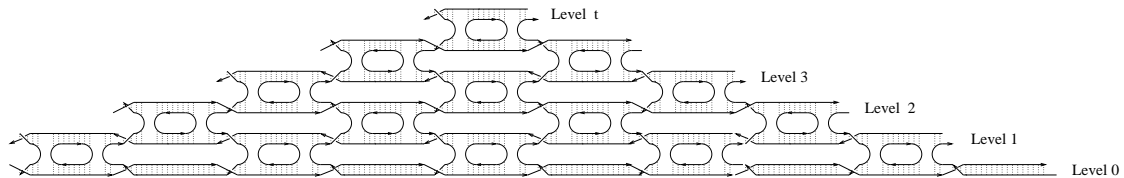
Figure 3.5: The DNA lattice resulting from a finite initial molecule. At the chosen annealing temperature, which is above the melting temperature for s base-pair annealing but below the melting temperature for $2s$ base-pair annealing, no more rule molecules can stably attach to this structure. However, if the bottom level (the initial molecule) were extended, then a larger triangle could form. s is the length of the sticky ends in the rule molecules.

Finally we ask, how can we access the output of the computation? This breaks down into two questions: How do we know *when* the computation is done? And *what* is on the tape at that point? There are many possible approaches to take; here we will merely sketch one. As mentioned above, we will consider the computation to be done when a special halting symbol is written on the tape.[9] In DNA, this corresponds to the special sticky end motif being incorporated into the lattice. When this occurs, the motif will be present as a double-stranded molecule for the first time, and this site

---

[9]At this point other parts of the tape will typically "not know" that the computation is done, so the lattice will continue to grow. However, it is also possible to design the cellular automaton such that all cells go into a special state to halt computation at the same time (the Firing Squad Problem, see *e.g.* Yunes (1994)), thereby allowing us to design linear pieces of DNA which fit into the gaps at the final level of the lattice, so that it cannot grow further. This may make extraction of the final tape configuration easier.

can be be chosen as the recognition domain for a binding protein[10], which could, for example, subsequently catalyze a phosphorescent reaction, turning the solution blue. To determine what is "on the tape" at this point, it is necessary to extract the single strand of DNA corresponding to the final level of the BCA. To do this, first add ligase to covalently bond all the annealed segments[11]. Then add resolvase to break all the crossover junctions[12]. Finally, heat to separate the strands, and use affinity purification to extract the strand containing the halting motif. Amplify and sequence that strand however you desire (*e.g.* via PCR and standard sequencing gels).

To summarize the model suggested here, a computation would proceed as follows.

1. First, express your problem via computer program. Convert that program into a (possibly nondeterministic) blocked cellular automaton.

2. Create small molecules (H-shaped and linear) which self-assemble to create the initial molecule (or initial molecules, if search over a FSA-generated set of strings is desired). Add ligase to strengthen the molecule.

3. Create small H-shaped molecules encoding the rule table for your program.

4. Mix the molecules created in steps 2 and 3 together in a test tube, and keep under precise conditions (temperature, salt concentrations) as the DNA lattice crystallizes.

5. When the solution turns blue, ligate, cut the crossovers, and extract the strand with the halting symbol.

6. Sequence the answer.

Analysis and Estimates. Will it work?

Let's begin the analysis optimistically. The above construction is just one implementation possible in a general class that might be called "crystal computation"[13]. In this class, we design a system where we can tailor-make the energy (and hence free energy) as a function of the configuration. We design it such that the lowest energy state (or in our case, the lowest free-energy state at a given temperature) uniquely represents the answer to our computation. This is closely related to the approach taken by Hopfield (1982) in his seminal work on neural networks. In our case the lowest energy configuration is one where every rule molecule has all four sticky ends bound. Given the presence of the initial molecule, this can only occur if the computation proceeds as desired.

The above analysis is a simplification that fails to take into consideration many aspects of the proposed implementation. For example, it completely ignores the dynamics involved; one simply anneals at a slow enough schedule, the argument goes, and the crystal is the result. Whereas in

---

[10]The protein must have an active bound form, and inactive unbound form. Furthermore, we must be sure it doesn't bind to rule molecules in the solution.

[11]It is a valid concern that ligase may not be able to bind to any but the outermost strands in a lattice. It may be better to reverse the order of the ligase and resolvase steps.

[12]Although a resolvase has been shown to cut crossovers in double-crossover molecules (Fu et al. 1994), it is unknown whether the enzyme will be functional on the inner strands in the lattice. However, the enzyme may be able to, at diminished speed, work from the edges in.

[13]It has been suggested that we shouldn't use the term "crystal", because it has a well-defined special meaning. At best, our constructions yield "pseudo-crystals", because any useful computation is aperiodic. We beg the reader to give us slack in using this term.

fact the crystallization proceeds at the edges only, according to kinetics that significantly influence the result.

Can a temperature be found such that two sticky ends bound is stable, while one sticky end bound is unstable? In other words, let T_0, T_1, and T_2 be the melting temperatures for a rule molecule fitting into a lattice slot where respectively 0, 1, and 2 of the sticky end pairings match. We want to keep the test tube at a temperature T such that $T_0 < T_1 < T < T_2$. This should be possible, but how large is the difference between T_1 and T_2? Although this is unknown for the particular molecules we use, we can get some idea by looking at what's known about linear DNA annealing. For example, under standard conditions 20 base-pair oligonucleotides (representing rule molecules with two length 10 sticky ends bound) melt at 70° C, while 14 base-pair oligonucleotides (representing rule molecules with only one length 10 sticky end bound, and the other matching partially) melt at 58° C (Wetmur 1991). $T = 65^\circ$ C would then discriminate the two cases. However, the analogy of rule molecules with two separate binding domains to variable-length oligonucleotides with continuous binding domains is questionable.

A definitive answer to "But will it work?" requires a chemist's knowledge and actual experiments. But we can immediately bring some more concerns to light. Since I do not have answers to them, I will merely mention them in passing. First, to read out an answer of more than one bit, our implementation requires ligating the rule molecules and cutting them with resolvase. It is not at all clear that, in the crowded confines of the DNA lattice, either ligase or resolvase will have room enough to perform its job[14]. Second, it is possible that, at a low rate, incorrect rules will be incorporated into the lattice. If this occurs, the computation is ruined. It is thus not clear at this time what yields of correct computation are to be expected, and whether a means could be devised to separate the good from the bad. It is additionally conceivable that stable structures form in the solution unconnected to the initial molecule. For example, four rules molecules could connect in a stable "diamond"; we might think that these complexes will only rarely be formed, because the intermediate steps are unstable (only one sticky end joins molecules), and for similar reasons they would grow slowly. However, they and other types of spurious connections and tangles could form, ruining the computation. A final concern is that there may be some systematic molecular stress or strain that comes into play when building a large crystal, and that beyond a certain size tearing would result. All these issues, and surely others, deserve more attention and study.

If for the moment we suppose that the implementation operates correctly, let us consider what advantage would be derived. Take the following with a bucket of salt: First, a small rule molecule (see Figure 3.6 for a close-up) consists of 50 base-pairs of DNA, sufficient for sticky ends of length 5, which gives us ≈ 10 symbols[15]. That's 33 K Dalton / rule molecule, with a size probably less than 20 x 44 x 85 Angstroms, for 3 bits / rule molecule.

Assessing speed is even more speculative. Suppose we perform a computation of a 10000-cell BCA with inactive boundary conditions, and compute for 10000 time steps. Suppose it takes 1 second for a rule to fit in when its slot is exposed. Since the 5000 slots are simultaneously exposed, all should be filled in approximately 1 second on average. This leads to a rough estimate of 3 hours for computing the 10000^2 cell lattice. Using 1kg of DNA, we could assemble 10^{19} rule molecules,

---

[14]If there is an angle between the plane of the lattice and a rule molecule which has just fit in place, then in our construction, an opposite angle is formed when a rule molecule fits into the subsequent layer. Consequently, the 2D lattice, rather than being perfectly planar, folds back and forth like a paper fan, which we call a "corrugated" lattice. The corrugated lattice exposes more of the double helix strands in each rule molecule, possibly making the strands more accessible to ligase but making the crossovers less accessible to resolvase.

[15]We optimistically require only 2 mismatches between sequences representing differing symbols. We also require the complement of a symbol's sequence does not code for a symbol, and that every code sequence has 3 C-G bonds and 2 A-T bonds, for more consistent melting temperatures.

34 $\overset{\circ}{\text{A}}$

f(x,y)

g(x,y)

44 $\overset{\circ}{\text{A}}$

x
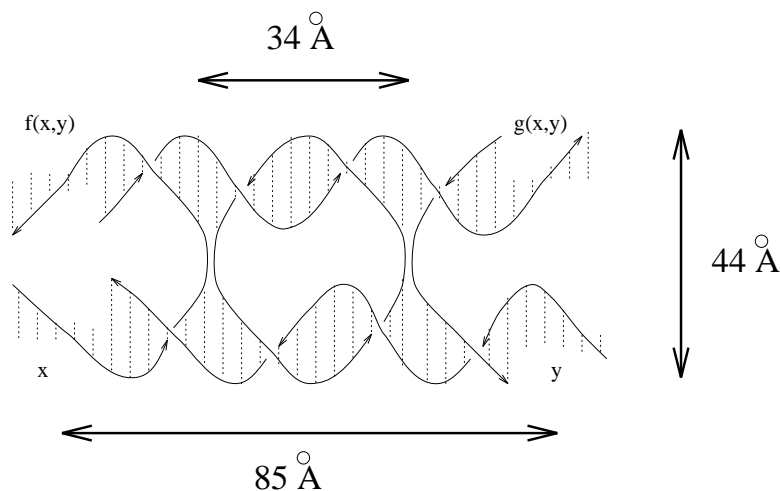
y

85 $\overset{\circ}{\text{A}}$

Figure 3.6: Detail of a small rule molecule. This is the smallest DAE/even style rule molecule possible. It has sticky ends of length 5, and internal region of length 10. Every base pair is shown.

that is, 10^{11} such calculations in parallel. That leads to a total of 10^{15} operations per second[16]. There is no lab work to be done during this the major stage in the computation. Of course time would also be required in the input and output stages.

Open Questions, Extensions, and Other Speculation.

In addition to the essential question of whether the ideas above can be made to work in the lab, there are many other issues to be investigated.

How energy-efficient is crystal computation? It is interesting to note that what might be called the computation proper (crystallizing the DNA lattice) theoretically requires arbitrarily little energy, as will be argued in the following sections. Of course, a great deal of energy may be used to heat the mixture up, to pulse the temperature to dissolve defects, or to apply other error-correcting mechanisms. Furthermore, the input and output stages require synthesis and analysis of DNA molecules, and thus also much energy. Our proposal is possibly the most nearly implementable example of the principle that computation is free, but input and output are costly (Bennett 1973).

Why use the DAE structure for rule molecules? Clearly the particular choice of molecule is not of intrinsic importance to the idea of this construction. The logical essence is to have an "H"-shaped molecule with four designable sticky ends. At its simplest, one could imagine making the "H" out of two chemically cross-linked strands of DNA (Figure 3.7a). Another alternative is the slightly larger single crossover Holliday junction. However, it is important for the construction of the lattice that the two linear pieces in the "H" be planar; Holliday junctions have been shown to prefer a (flexible) $60°$ skew angle (Eis and Millar 1993). The chemically linked strands imagined above have not yet been characterized. The reason we propose the large double crossover molecules[17] is that they have already been characterized in the lab and are thought to be rigid

---

[16]This compares to 300 GFLOPS ($\approx 10^{14}$ basic operations per second) attainable by the best modern supercomputers, *e.g.* a 7000 processor Intel Paragon. Of course, the "operations" we compare are apples and oranges.

[17]Ned Seeman suggested we consider double crossover molecules as an improvement over the more awkward branched junction constructions we were originally considering.
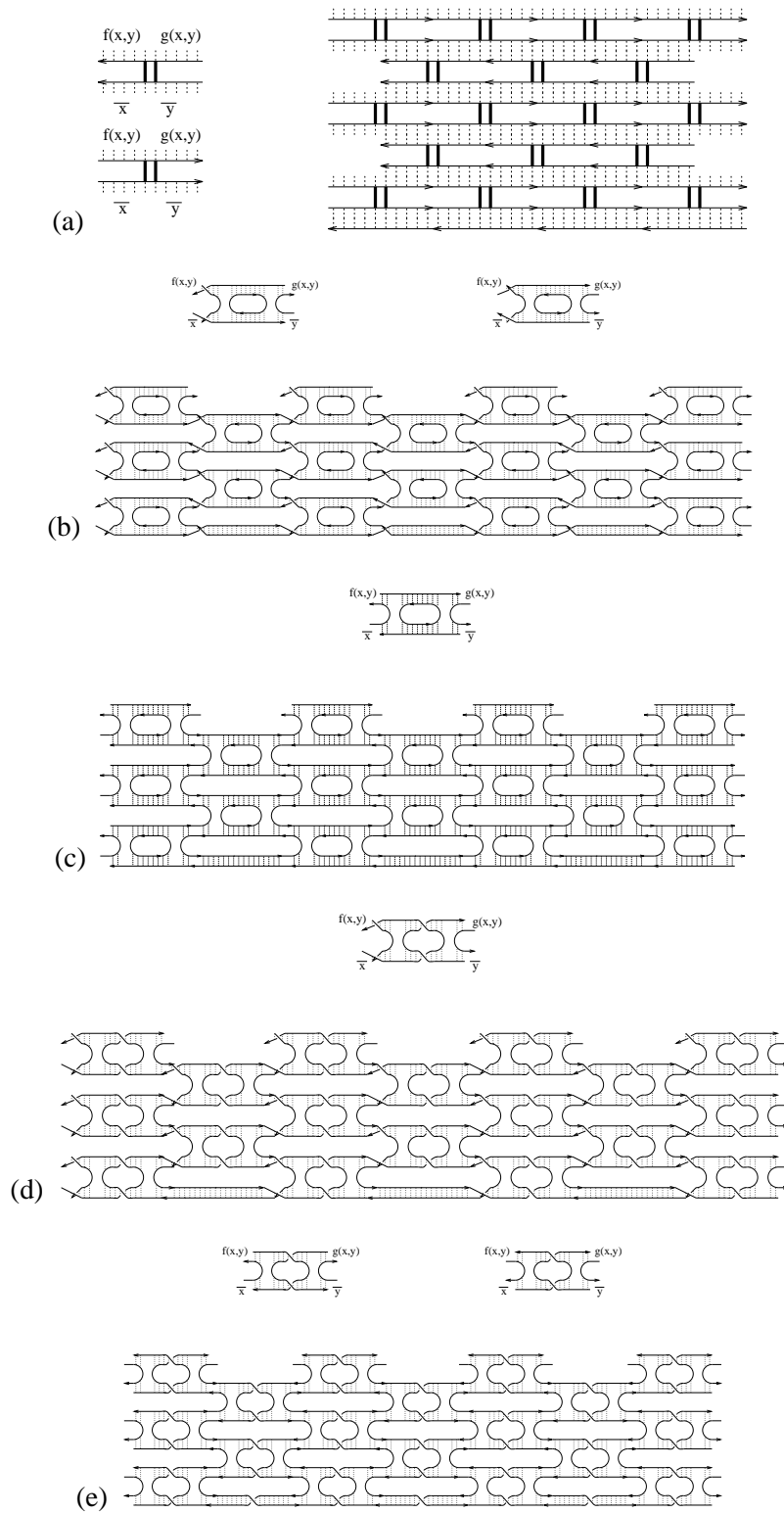
37

Figure 3.7: Alternative Topologies for 2D Lattice. (a) Rule molecules based on cross-linked DNA. (b) DAE rule molecules with an odd number of half-turns between junctions on adjacent molecules. (c) DAE rule molecules with even-length spacing. (d) DAO rule molecules with odd-length spacing. (e) DAO rule molecules with even-length spacing.

(which may help prevent tangled lattices) and planar (Fu and Seeman 1993). We chose DAE in preference to other topological variants of double crossover molecules, such as DAO, because the topology of the rule molecule leads to a different "weave" of DNA strands in the lattice (Figure 3.7bcde). We prefer to have a single strand which, if covalently linked, runs along an entire level of the lattice, thus encoding the BCA state for that time step.

Why a 1D BCA? Why not build a 3D lattice to simulate a 2D BCA? We started with 1D BCA because they can be immediately explored used existing DNA technology. Two dimensions offers several advantages, however, such easier design of efficient computations. Perhaps more importantly, in higher dimensions it becomes easier to design error-tolerant rules (Gacs and Reif 1988); intuitively, point defects in 2D can be filled-in from adjacent correctly-computed cells, while in 1D a point defect severs communication between the left and right side. *Open question: Can the DNA rule molecules be modified so as to build 3D DNA lattices?* Speculatively, one could propose a variant of the double crossover Holliday junction, the "multiple strand double crossover junction" (Figure 3.8), as a means to implement the read-4, write-4 operation required by 2D blocked cellular automata (see e.g Toffoli and Margolus (1987), Ch. 12). Unfortunately, the proposed building-block molecule has not yet been synthesized.
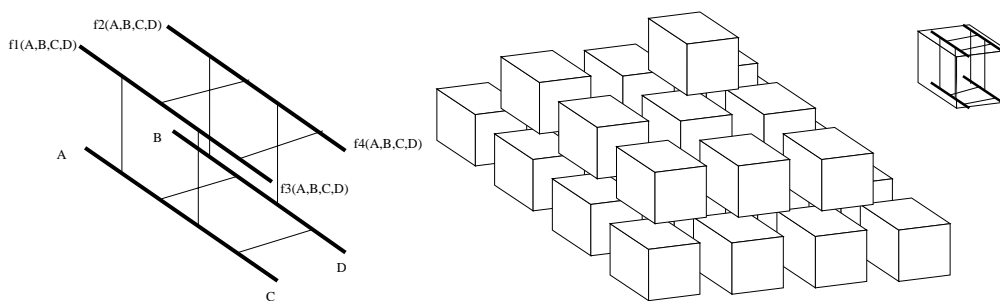
Figure 3.8: A possible 3D lattice of DNA for simulating 2D BCA. Four DNA double helices may be bound together by crossover junctions (left). Sticky ends determine 2D BCA rules as the rule molecules assemble in an alternative cubic lattice (right).

Potential uses in nanotechnology. Here we have suggested an approach to molecular computation via programmable self-assembly. Programmable self-assembly may have other applications. *Open question: Can cellular automata generated lattices be used to define ultra-high resolution electronic circuits?* One possibility, along the lines investigated by Robinson and Seeman (1987), would be to conjugate nano-wire onto individual rule molecules, such that when the rule molecules fit together, an electrical circuit is formed. This proposal differs from Robinson and Seeman's suggestion in that whereas they envisioned a periodic lattice of identical memory cells, we suggest that cellular automata rules could be used to build more complicated circuits, either in 2D or 3D.

Why use DNA at all? The principle of computing via crystallization is not restricted to DNA. *Open question[18]: Can non-DNA-based molecules could be used to design desired computations carried out on the surface of a growing crystal?*

---

[18]*Suggested by Stuart Kauffman, private communication.*

3.1.4 Comparison with Other Approaches

Perhaps the most practical suggestion for universal computation via DNA is that of Boneh et al. (1996a). Their approach makes straightforward use of well understood laboratory techniques for manipulating DNA. They are able to simulate nondeterministic boolean circuits, which seems very efficient for some calculations, and which gives them universal computational ability. Because circuits allow non-local interactions of variable, circuits can be very compact. However, it should be pointed out that the computation requires a lab technician to sequence operations on multiple test tubes; the logic of the program being computed is external to the DNA, which is used as a memory. Small scale computations could be immediately attempted with reasonable chance for success; however due to the weakness of single-stranded DNA and other factors, it is not clear how this approach will scale.

Other authors have proposed DNA implementations of Turing Machines directly (*e.g.* Beaver (1996), Smith (1996), Rothemund (1996)). The approaches vary from using PCR to relying on restriction enzymes. These approaches show promise, although the reliability and efficiency of the steps is unclear. Furthermore, single-tape, single-head Turing Machines are particularly cumbersome logically; circuits will typically compute the same function in many fewer steps (and single steps take comparable time in both systems – on the order of hours!). In short, although they are of theoretical interest, it is unlikely that anyone will actually go into the lab and solve problems this way.

Our hypothetical cellular automaton implementation differs in a number of ways: First and foremost, our proposal is a "one-pot" reaction. Dump in the rule molecules encoding your problem, and all the logic of the computation is carried out autonomously. No lab work is involved. Furthermore, in addition to running a massive number of computations in parallel, each cellular automaton performs its own computation in parallel – thus fully exploiting the parallelism available. The major and significant drawback of our proposal is that it makes use of chemistry which is not yet fully understood, and thus going into the lab to do a computation this way would be a real technical challenge.

The main conclusion of this paper is that annealing and ligation *alone* may be sufficient for universal "one-pot" DNA computation. Whether the particular scheme envisioned here can be made to work in the lab is a matter for further research. In any case, it is clear that better experimental characterization of the chemistry of annealing is required, and may open up new possibilities for DNA based computation.

3.2   Graph-Theoretic Models of DNA Self-Assembly

Abstract[19]   In this paper we examine the computational capabilities inherent in the hybridization of DNA molecules. First we consider theoretical models, and show that the self-assembly of oligonucleotides into linear duplex DNA can only generate sets of sequences equivalent to regular languages. If branched DNA is used for self-assembly of dendrimer structures, only sets of sequences equivalent to context-free languages can be achieved. In contrast, the self-assembly of double crossover molecules into two dimensional sheets

---

[19]Results in this section also appear in Winfree et al. (in press). Thanks to Dan Abrahams-Gessel for suggesting the context-free grammar result.

or three dimensional solids is theoretically capable of universal computation. The proof relies on a very direct simulation of a universal class of cellular automata.

A fundamental property of DNA is that, under the right conditions, Watson-Crick complementary regions of single-stranded DNA will hybridize and form a double helical structure. This property, in vitro and in vivo, can lead DNA to assume a remarkable diversity of geometric forms[20]. Under certain simplifying conditions, the behavior of hybridization is sufficiently predictable to be considered as a computational primitive; i.e., a function from initial oligonucleotides to final supramolecular structures is computed. The computational aspects of self-assembly were exploited for the first time in Adleman (1994), where linear self-assembly was used as a step in solving the Hamiltonian Path Problem. When the self-assembly of tree-like structures takes place, due to the presence of branched junctions, a slightly more powerful computation results. We review a two dimensional generalization capable of universal computation, as suggested in Winfree (1996b), and also suggest a concrete three dimensional self-assembly process.

In order to understand the computational implications of DNA hybridization, we will first consider a highly abstracted mathematical model. The physical system we would like to model can be described as follows:

> Synthesize several sequences of DNA. Mix the DNA together in solution. Heat it up and slowly cool it down, allowing complexes of DNA to form. Chemically or enzymatically ligate adjacent strands. Denature the DNA again, and ask, what single-stranded DNA sequences are now present in the solution?

A proper answer to this question is beyond our capability, and realistically detailed models might not be enlightening regarding the logical essence of self-assembly. We therefore investigate very simple models, which, nonetheless, are sufficiently realistic that translation into real world scenarios should be direct. We will consider a number of properties which DNA self-assembly may be postulated to obey, and we will analyze the computational capability and the limits of any self-assembly process which obeys those properties.

Informally, the properties we consider are:

1. **Constant Temperature.** The number of base-pairs required for the stability of DNA complexes does not change during the course of the self-assembly. We thus don't consider annealing, where at high temperatures only long regions will hybridize but later at lower temperatures even short regions can hybridize, but rather we model a "constant temperature" process.

2. **Perfect Watson-Crick Complementarity.** Hybridization only occurs between sequences with perfect Watson-Crick complementarity. Hybridization of mismatching sequences, or that which creates bubbles, branched junctions, triple helices, and other unusual structures, is not considered.

3. **Permanent Binary Events.** All self-assembly interactions occur between two complexes at a time, and no more. These interactions are exclusively hybridizations, joining two complexes together. Furthermore, in the model once two complexes join, they never dissociate.

---

[20]In vivo, not only is there single-stranded and double-stranded DNA, but branched junctions are formed during recombination, and trypanosomes maintain complex networks of circular DNA within which RNA editing occurs.

4. **No Intramolecular Events.** A DNA complex which has self-assembled will not interact with itself, for example by cyclizing. Note, however, that some physically intramolecular interactions can be modeled as a part of a binary event, as discussed below.

5. **Single vs Multiple Binding Regions per Event.** We will consider two cases: either (a) each binary hybridization event creates a single contiguous Watson-Crick region, else (b) the binary events may result in the formation of several physically separated hybridized regions between the two complexes. The latter case is meant to model physical situations where an intermolecular hybridization is immediately followed by an intramolecular hybridization. The case we are interested in is discussed in Section 3.2.5 (see Figure 3.15).

6. **Specified Classes of Initial Complexes.** Because of our constant-temperature assumption, it becomes useful to assume that some complexes have already formed prior to the stage of self-assembly which we will consider. Later in the paper, we will consider initial complexes which consist of (a) oligonucleotides, (b) duplex DNA with sticky ends, (c) hairpins with sticky ends, (d) three-armed junctions with sticky ends, (e) double crossover molecules with hairpins and sticky ends, and (f) arbitrary complexes.

Properties (1), (2) and (3) are used primarily for logical simplicity. If Property (4) were changed to allow intramolecular events, it is possible that some of our results would be slightly modified. We will analyze how our results change under different choices for Properties (5) and (6). In Section 3.2.5, we impose an additional property in order to incorporate geometrical considerations for lattice self-assembly.

3.2.1 Language Theory and Grammars

Before we present our model of DNA self-assembly, we should comment on what it means to compute by self-assembly. As mentioned above, the typical case is that one starts with a small variety of synthesized oligonucleotides, and one ends with great variety of self-assembled strands. The resulting strands are not random; they have certain properties that derive from being formed from the original oligonucleotides according to certain rules of hybridization.

An analogous situation arises in formal language theory, which has been well understood for many years. There, rather than test tubes of strands, one is interested in sets of symbolic strings, and in methods of generating them. We will sketch the basics here; for a full development see Ginsburg (1966).

An *alphabet* is a finite set of symbols, for example $\{A, C, G, T\}$ or $\{0, 1\}$ or $\{x, y, z, (,), +, *\}$. A *string* over an alphabet is a finite sequence of symbols from the given alphabet, for example $TATAA$ or 101011 or $(x + y) * z$. A *language* is a well-defined, possibly infinite set of strings, for example { *all strings over* $\{C, T\}$ *of length* 70 } or { *all prime numbers, written in binary* } or { *all well-formed formulas over* $\{x, y, z, (,), +, -\}$ }.

Although one cannot write down each and every string in an infinite language, one can ask the membership question: is string x in language $\mathcal{L}$? Note that if the language $\mathcal{L}$ contains all bit strings x for which function $f(x) = 1$, the the membership question is equivalent to boolean function evaluation. The membership question may be harder or easier to answer, depending on x and $\mathcal{L}$. Formal language theory goes to great pains to classify languages according to how fancy the computer must be to answer the membership problem. We sketch the fundamental result due to Noam Chomsky, known as the language hierarchy. This requires formalizing the specification of languages by generative rules.

A *rewriting rule* $x \to y$, where x and y are strings, specifies that a string $s = axb$ can be rewritten to produce the new string $s' = ayb$. A *grammar* G is a collection of rewriting rules together with a division of the alphabet into two groups: *terminal symbols* and *nonterminal symbols*, where only nonterminals appear on the left hand side of rewriting rules. Each grammar uniquely defines a language $\mathcal{L}_G$ as follows: the string of terminals s is in $\mathcal{L}_G$ iff it can be obtained from the special nonterminal S by the repeated application of rewriting rules in some order (called a *derivation*).

Grammars may be classified by what kinds of rules they use. We give examples of the three main classes below:

Regular grammars use rules of the form $A \to pB$ and $A \to p$ where A and B are nonterminal symbols and p is a string of terminals. Languages generated by regular grammars are called regular languages. For example, consider the regular grammar $G_E = \{S \to 0S, S \to 1T, S \to 0, T \to 0T, T \to 1S, T \to 1\}$ where 0 and 1 are terminals. This grammar gives rise to all bit strings with an even number of 1's. $101011 \in \mathcal{L}_{G_E}$ because $S \to 1T \to 10T \to 101S \to 1010S \to 10101T \to 101011$. Note that during the derivation we always have a single nonterminal at the right, where all the action takes place. Despite their apparent simplicity, regular languages have found extensive use in pure and applied computer science, perhaps because their membership question can always be answered by an exceedingly simple abstract computer known as a finite state machine.

Context-free grammars use rules of the form $A \to P$ where again A is a nonterminal symbol, but now P is an arbitrary string of terminals and nonterminals. Languages generated by context-free grammars are called context-free languages. Consider the grammar $G_F = \{S \to S + S, S \to M, M \to M * M, M \to (S), M \to x, M \to y, M \to z\}$ where the terminals are $\{x, y, z, (,), +, *\}$. This grammar gives rise to well-formed formulas. $(x + y) * z \in \mathcal{L}_{G_F}$ because $S \to M \to M * M \to M * z \to (S) * z \to (S + S) * z \to (S + M) * z \to (S + y) * z \to (M + y) * z \to (x + y) * z$. Note that whereas it is impossible to generate regular languages whose strings all have long-range structure, one can generate long-range "nested" structure in a context-free language – for example, every parenthesis must be matched in the formulas above. Context-free languages include regular languages. The membership question for context-free languages can be answered by a slightly more complex machine known as a nondeterministic pushdown automaton.

Unrestricted grammars use rules of the form $A \to P$ where now A may be an arbitrary strings of nonterminals, and P is an arbitrary string of terminals and nonterminals. Languages generated by unrestricted grammars are called *recursively enumerable* languages because they include every language which can be generated (enumerated) by any computational process (recursion). Recursively enumerable languages include context-free languages, regular languages, and much more. They are as fancy as you can get. A very simple example: consider the alphabet $\{S, L, R, \overleftarrow{B}, \overrightarrow{B}, \overleftarrow{W}, \overrightarrow{W}, 0, 1\}$ and the grammar $G_P = \{S \to 1, S \to LR, L \to L\overrightarrow{B}, L \to 1, R \to \overleftarrow{B}R, R \to 1, \overrightarrow{B}\overleftarrow{B} \to \overrightarrow{W}\overleftarrow{W}, \overrightarrow{B}\overleftarrow{B} \to 0, \overrightarrow{B}\overleftarrow{W} \to \overrightarrow{B}\overleftarrow{B}, \overrightarrow{B}\overleftarrow{W} \to 1, \overrightarrow{W}\overleftarrow{B} \to \overrightarrow{B}\overleftarrow{B}, \overrightarrow{W}\overleftarrow{B} \to 1, \overrightarrow{W}\overleftarrow{W} \to \overrightarrow{W}\overleftarrow{W}, \overrightarrow{W}\overleftarrow{W} \to 0\}$ where the terminals are 0 and 1. This gives rise to the rows of Pascal's triangle mod 2. The third row $101 \in \mathcal{L}_{G_P}$ because $S \to LR \to L\overrightarrow{B}R \to L\overrightarrow{B}\overleftarrow{B}R \to 1\overrightarrow{B}\overleftarrow{B}R \to 10R \to 101$. Later, we will make use of a subclass of unrestricted grammars equivalent to blocked cellular automata, which generalize the example and which are still capable of generating all recursively enumerable languages; that is, they are *universal*. A surprising consequence of universality is that the membership

question for recursively enumerable languages is sometimes impossible to answer!

3.2.2 DNA Complexes and Self-Assembly Rules

Grammars turn out to have a close relationship to the self-assembly models we discuss here. However, to make this relationship precise, we define our model formally.

A *DNA complex*[21] is a connected directed graph with vertices labeled from $\{A, C, G, T\}$, edges labeled from $\{backbone, basepair\}$, with at most one incoming and one outgoing edge of each type at each node (thus at most four incident edges total), and where for every base pair edge $x \to y$ there is a reciprocal basepair edge $y \to x$. Furthermore, all base-pairing in a DNA complex must be *Watson-Crick*, that is, every basepair edge must be within a subgraph isomorphic to one of the 10 given in Figure 3.9a.
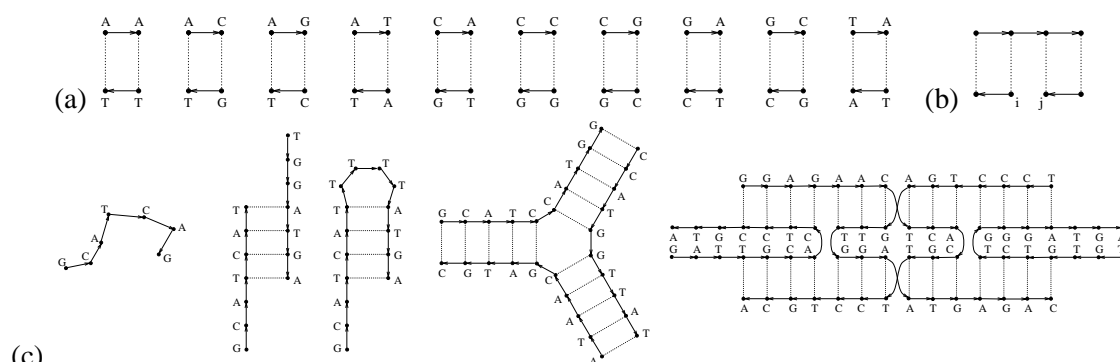

Figure 3.9: Some DNA complexes. Solid lines represent backbone edges; each dotted line represents a pair of reciprocal basepair edges. (a) The 10 Watson-Crick subgraphs. (b) The valid ligation site. (c) A strand, a duplex with sticky ends, a hairpin with a sticky end, a 3-armed branched junction, and a DAO double crossover (DX) unit with sticky ends.

A DNA complex (just *complex* for short) represents several DNA polynucleotides bound together by Watson-Crick hybridization. Note that this representation supports a rich variety of DNA structures, but structures such as triple helices are missing; similarly, it is lacking notions of geometry and topological linking. Also, we must be careful because it is possibly to specify physically impossibly structures.

It will be useful to introduce a few examples of DNA complexes, shown in Figure 3.9c. A *strand* consists of a chain of backbone-connected nodes, with no basepair edges. Strands may be either linear or circular. A *duplex* consists of two strands with contiguous basepair edges between them. A duplex may optionally have a *sticky-end* on either end. An n-*armed (branched) junction* consists of n duplex arms arranged around a central point. A *double crossover unit* (DX unit) consists of two adjacent duplexes with two points of strand exchange[22]. For formal reasons, the empty graph ϵ is a DNA complex.

We now define some operations on complexes. In our model, hybridization is indicated by $C_1 +_B C_2 = C_3$, where $+_B$ denotes the formation of basepair edges B between nodes of C_1 and

---

[21]Similar to the *cluster* in Beaver (1995).

[22]Real DX molecules (Fu and Seeman 1993) come in a number of geometric varieties (we use "DAO" here), each of which put constraints on the symmetry and the number of nucleotides between crossover points. We ignore these constraints in the theoretical section.

nodes of C_2. If the graph consisting of both C_1 and C_2 and the edges B is a DNA complex, then C_3 is that graph; else $C_3 = \epsilon$ (for example, if a new edge joins two T's). The hybridization operation will be used to describe self-assembly, below.

To analyze the complexes present after self-assembly, we introduce two other operations based on ligation and denaturing. $C' = ligate(C)$ is obtained by adding a backbone edge from node j to node i in every occurrence of the subgraph shown in Figure 3.9b, so long as nodes i and j have no other incident backbone edges.

To model the denaturing of a complex, we define $\{C_i'\} = denature(C)$ to be the set of all *strands* in C, i.e., each C_i is a backbone-connected component of C (with no basepair edges). Note that if C contains topologically linked circular strands, then *denature* will "magically" unlink them from each other[23].

In analogy to formal language theory, we define a *language of DNA complexes* to be a well-defined, possibly infinite set of DNA complexes. We can generate a language of complexes $\mathcal{L}_{R,A}$ by applying *self-assembly rules* R to an initial language A, usually finite[24]. The rules R specify which hybridizations $C_1 +_B C_2 = C_3$ are allowed. Let $\hat{\mathcal{L}}_{R,A}$ be the transitive closure of A under all allowed hybridizations. In other words, (a) $A \subset \hat{\mathcal{L}}_{R,A}$, (b) if $C_1, C_2 \in \hat{\mathcal{L}}_{R,A}$ and $C_1 +_B C_2 = C_3$ is allowed, then $C_3 \in \hat{\mathcal{L}}_{R,A}$, and (c) no other complexes are in $\hat{\mathcal{L}}_{R,A}$. Now let $\mathcal{L}_{R,A} \subset \hat{\mathcal{L}}_{R,A}$ consist of those complexes for which no further hybridization is allowed; these are called *terminal complexes*. Loosely, $\mathcal{L}_{R,A}$ is meant to model the DNA structures which would form given an infinite volume of DNA and infinite time, presuming that only the hybridizations allowed by R are physically relevant, and ignoring transient structures.

We will be especially interested in the self-assembly rules[25] R_1^T which allow $C_1 +_B C_2 = C_3 \neq \epsilon$ iff (1) the subgraph of C_3 induced by B contains exactly two T-mer (or longer) strands and (2) at most two edges lead to or exit from this subgraph. Thus, R_1^T allows only hybridization of sufficiently long sticky-ends, as illustrated in Figure 3.10.
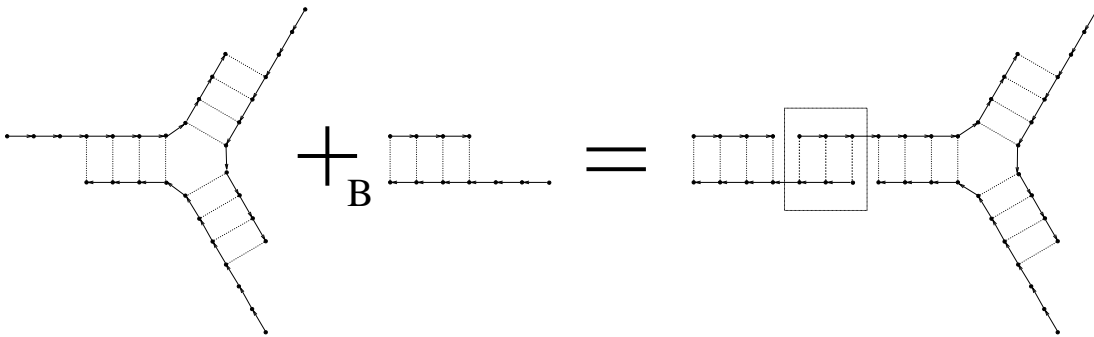
Figure 3.10: A hybridization $C_1 +_B C_2 = C_3$ allowed by R_1^3. The edges of B are emphasized in C_3, and the subgraph induced by B has a dotted box around it.

Both *ligate* and *denature* can be generalized to set operations by applying the operation to each complex in the original set, and taking the union of all complexes that result. Since single-stranded DNA can be identified with its sequence, written 5' $\rightarrow$ 3', we can consider *denature*

---

[23]Circular strands are not necessary in our constructions, but they must be considered in Theorem 2(2).

[24]Logicians may think of A as "axioms" while R may be thought of as "inference rules".

[25]The subscript "1" is used because these rules give rise to essentially one-dimensional complexes.

to be a function from sets of complexes to sets of strings over $\{A, C, G, T, \cdot\}$, where $\cdot$ is used to indicate a circular DNA strand.

Finally, we note that to represent strings in alphabets Σ other than $\{A, C, G, T\}$, we may use a *prefix-free codebook* $\mathcal{C}$ which assigns to each symbol σ in Σ a string $\mathcal{C}_\sigma$ over $\{A, C, G, T\}$ such that no string is a prefix of another string. A DNA sequence $s = s_1 s_2 \ldots s_n$ can then be translated into a string $\mathcal{C}(s)$ over Σ by scanning through s from left to right: if s_i begins a subsequence of s which exactly matches some $\mathcal{C}_\sigma$, then s_i is replaced by σ, else s_i is erased; then s_{i+1} is processed, and so forth. For example, if $\Sigma = \{0, 1\}$, $\mathcal{C}_0 = CAG$, and $\mathcal{C}_1 = CTC$, then $\mathcal{C}(AAACTCTCAGTCAG) = 1100$.

In summary, given a finite set of complexes A, self-assembly rules R, and codebook $\mathcal{C}$, we can obtain a language of complexes $\mathcal{L}_{R,A}$ as well as a language of strings

$$\mathcal{L}_{R,A,\mathcal{C}} = \mathcal{C}(denature(ligate(\mathcal{L}_{R,A}))).$$

We now turn to our results. The theorems are stated, explained, and examples are given. Full proofs will appear elsewhere.

3.2.3 Linear Self-Assembly is Equivalent to Regular Languages

In this section we address the question of what can be computed by the self-assembly DNA which obeys Properties (1-4), (5a), and (6a) or (6b). This is the familiar case of the self-assembly of long duplex DNA from many small oligonucleotides or sticky-ended fragments. That is, self-assembly begins with oligonucleotides or duplex DNA with sticky ends, and proceeds at a constant temperature, allowing only permanent binary events with a single perfectly complementary hybridization site and no intramolecular hybridization. We make this question precise in our model by asking, what languages of strings $\mathcal{L}$ can be achieved as $\mathcal{L}_{R_1^T,A,\mathcal{C}}$ for some choice of $T, \mathcal{C}$, and A where A contains only linear duplex complexes?

The following[26] can be proved by construction:

Theorem 1. (1) For all regular languages $\mathcal{L}$, there exists a positive integer T, a codebook $\mathcal{C}$, and a set of linear duplexes A such that $\mathcal{L} = \mathcal{L}_{R_1^T,A,\mathcal{C}}$. (2) For all positive integers T, codebooks $\mathcal{C}$, and sets of linear duplexes A, $\mathcal{L}_{R_1^T,A,\mathcal{C}}$ is a regular language.

We will sketch the construction used in the proof of (1) – see Figure 3.11 for an example. Consider a regular grammar G for $\mathcal{L}$. We design sufficiently dissimilar sequences S_i (we call their Watson-Crick complements S_i') for all the terminal and nonterminal symbols in G. For each rule $A \to p_1 \ldots p_n B$, we design a duplex with a sticky end S_A', and internal duplex region $S_{p_1} \ldots S_{p_n}$, and a sticky end S_B if B is present. We also design a duplex with one blunt end and a sticky end S_S, to represent the start symbol S. These duplexes make up the initial set of complexes A. T is chosen to be the length of the nonterminal sequences S_i. After self-assembly, the terminal complexes in $\mathcal{L}_{R_1^T,A}$ will correspond to derivations in G. After ligation, each complex will be a blunt-ended duplex whose sequence consists of terminal sequences interspersed with nonterminal sequences. A codebook with $\mathcal{C}_i = S_i$ for each terminal symbol i will "erase" the nonterminal sequences; thus $\mathcal{L}_{R_1^T,A,\mathcal{C}}$ will be exactly $\mathcal{L}$. $\square$

A sketch of the proof of (2) is as follows: we construct a regular grammar G which generates exactly the strands in $denature(ligate(\mathcal{L}_{R_1^T,A}))$. This requires creating a nonterminal symbol for each sticky end of a duplex in A, and considering all (finitely many) T-or-more base overlaps of these sticky-ends; a grammar rule is provided for each such interaction. Care must be taken

---
[26]We note that this theorem still holds when "duplexes" is replaced by "strands".
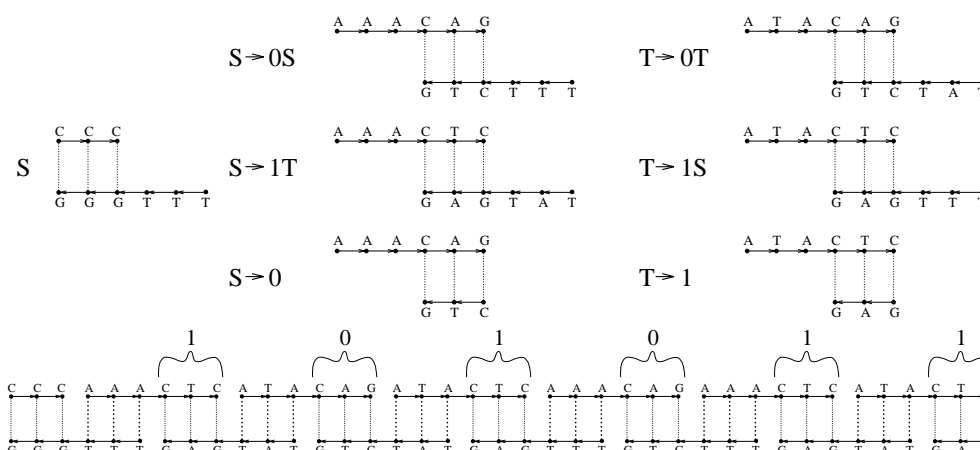
Figure 3.11: The initial complexes A corresponding to the regular grammar G_E, and an example derivation. Note that the self-assembly of the derivation could have occurred in any order. Subsequent ligation and denaturing will produce two strands (top and bottom) from this terminal complex. The codebook $\mathcal{C}$ defines $\mathcal{C}_0 = CAG$ and $\mathcal{C}_1 = CTC$. We use R_1^3.

for gaps and for sticky ends which have no interactions – both lead to termination of the strand sequence, and may require a rule using the start symbol S. Translation by the codebook can be effected by applying a nondeterministic finite state transducer (Ginsburg 1966) to $\mathcal{L}_G$, yielding a regular language equal to $\mathcal{L}_{R_1^T, A, \mathcal{C}}$. $\square$

Thus, our model for linear self-assembly does not permit very interesting computations. It should be emphasized that simple extensions might allow for more complex computations. For example, suppose hairpins appear in A in addition to duplexes. Then, for example, we could replace the duplex for S (Figure 3.11) by the hairpin , and change the codes for 0 and 1 to the Watson-Crick palindromes $CCGG$ and $CGCG$. Now both the top and bottom strands code the 0 and 1 sequences; furthermore, after ligation the top and bottom strands are joined together by the hairpin. Consequently, we generate the set of all palindromes in which the number of ones is a multiple of four – which is not a regular language! How far can we push this idea?

3.2.4 Dendrimer Self-Assembly is Equivalent to Context-Free Languages

Dan Abrahams-Gessel pointed out to me that dendrimer self-assembly looks formally identical to context-free grammars. This observation translates very nicely into DNA self-assembly of branched junctions into tree-like complexes. Therefore, in this section we address the question of what can be computed by the self-assembly of DNA which obeys Properties (1-4), (5a), and (6b-d). That is, self-assembly begins with duplexes, hairpins, and 3-armed junctions with sticky ends, and proceeds at a constant temperature, allowing only permanent binary events with a single perfectly complementary hybridization site and no intramolecular hybridization. We note that this form of self-assembly has not been widely studied in the lab, and that full self-assembly would be limited not only by material but also by geometric (steric) interference and volumetric constraints[27]. Nonetheless, our abstract model allows us to ask the following precise question:

---

[27]Consider a tree which branches at every opportunity. It has 2^n nodes within n steps of the center; but the volume of space within n steps grows only as n^3.

what languages of strings $\mathcal{L}$ can be achieved as $\mathcal{L}_{R_1^T,A,\mathcal{C}}$ for some choice of T, $\mathcal{C}$, and A where A contains only duplexes, hairpins, and 3-armed junctions?

An extra complication that immediately arises is the possibility that circular strands may form. Recall our convention that *denature* returns "dotted" sequences to represent circular strands, but didn't specify which permutation of the circle to use. It becomes convenient to work with equivalence classes of sequences, where $\cdot S \cong \cdot T$ if the sequences S and T are circular permutations of one another. Languages $\mathcal{L}_1$ and $\mathcal{L}_2$ are deemed *equivalent* if for every sequence S in one language, there is an identical or equivalent sequence T in the other language.

The following[28] can be proved by construction:

Theorem 2. (1) For all context-free languages $\mathcal{L}$, there exists a positive integer T, a codebook $\mathcal{C}$, and a set of duplexes, hairpins, and 3-armed junctions A such that $\mathcal{L} = \mathcal{L}_{R_1^T,A,\mathcal{C}}$. (2) For all positive integers T, codebooks $\mathcal{C}$, and sets of duplexes, hairpins, and 3-armed junctions A, $\mathcal{L}_{R_1^T,A,\mathcal{C}}$ is equivalent to a context-free language.
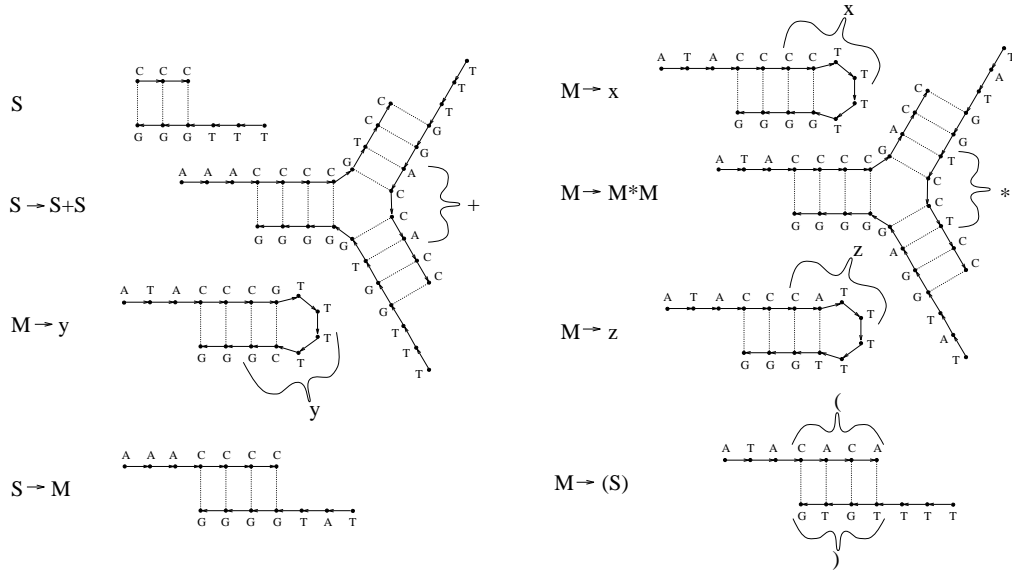
Figure 3.12: The initial complexes A corresponding to the regular grammar G_F. The codebook $\mathcal{C}$ defines $\mathcal{C}_x = CCTT, \mathcal{C}_y = TTCG, \mathcal{C}_z = CATT, \mathcal{C}_(= CACA, \mathcal{C}_) = TGTG, \mathcal{C}_+ = ACCA$, and $\mathcal{C}_* = TCCT$. We use R_1^3.

We will sketch the construction used in the proof of (1) – see Figure 3.12 and 3.13 for an example. The construction is similar to that in Theorem 1. Consider a context-free grammar G for $\mathcal{L}$. Note that there is an equivalent grammar $\hat{G}$ which uses rewriting rules of the form $A \to pBqCr$ where p, q, and r are (possibly null) strings of terminal symbols, and A, B, and C are single nonterminal symbols (or null). Again, we design sufficiently dissimilar sequences S for all the terminal and nonterminal symbols used in $\hat{G}$. For rules of the form $A \to pB$ or $A \to Bp$ (B not null), we design a duplex as before. For rules of the form $A \to p$, we design a hairpin with the sequences for p in the stem. We design a 3-armed junction for each rule of the form $A \to pBqCr$ (B and C not null); it has sticky ends for S_A, S_B, and S_C, and the sequences for p, q, and r are

---

[28]We note that this theorem still holds when "duplexes, hairpins, and 3-armed junctions" is replaced by simply "complexes". That is to say, this is a fully general theorem for self-assembly under R_d^T.

placed on the arms. As before, we design a blunt-ended duplex for the start symbol S. These complexes make up the initial set of complexes A. As before, at the appropriate "temperature" T, the terminal complexes will correspond to derivations in $\hat{G}$, and ligation will convert each complex into a single strand which encodes the derivation. Processing with the codebook for the terminal symbols will "erase" the nonterminal sequences, and $\mathcal{L}_{R_1^T, A, \mathcal{C}}$ will be exactly $\mathcal{L}$. $\square$
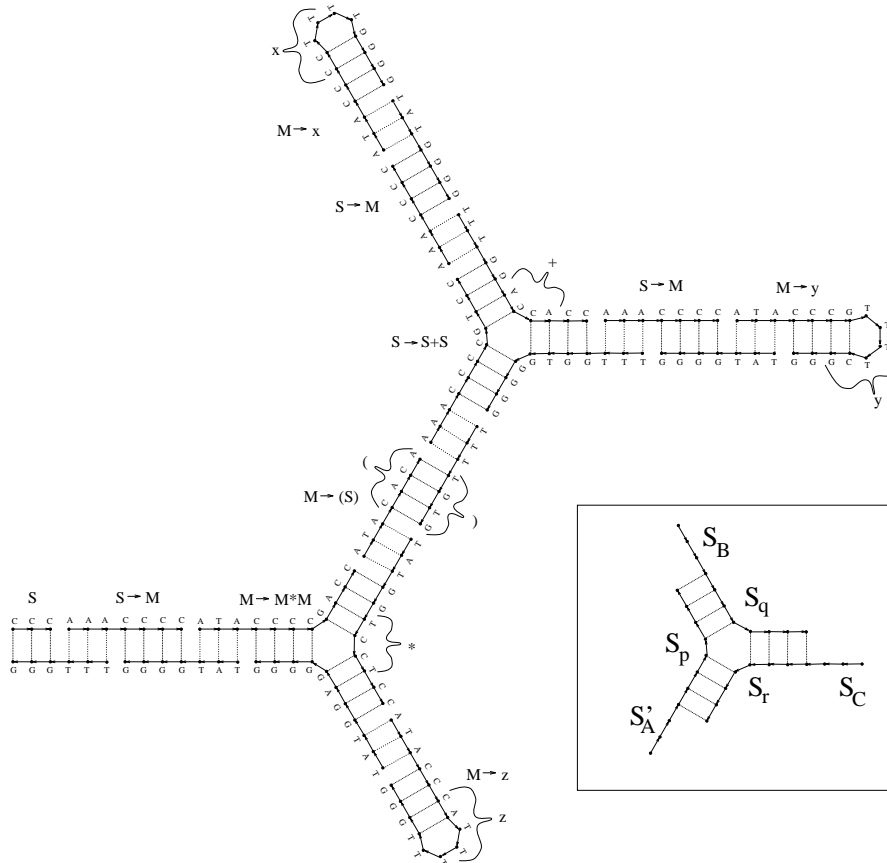
Figure 3.13: An example derivation by self-assembly of the complexes A corresponding to the regular grammar G_F. Note that the self-assembly of the derivation could have occurred in any order, using R_1^3. Subsequent ligation will produce a single strand from this terminal complex. Inset: the three-armed junction corresponding to the generic rewriting rule $A \to pBqCr$.

The proof of (2) also follows the form of the proof of Theorem 1, only now we construct a context-free grammar G which, loosely speaking, generates sequences corresponding to backbone paths through complexes in $ligate(\mathcal{L}_{R_1^T, A})$, where gaps are filled in with the symbol $\diamond$, and where several (but not necessarily all) permutations of each circular strand are given using $\cdot$. This language is then passed through a nondeterministic transducer which returns the strand sequences in $\{A, C, G, T\}$ and circular strand sequences in $\{A, C, G, T, \cdot\}$. As before, the final strings are produced by another nondeterministic transducer, which this time translates using the codebook. Thus the final language is context-free, and is equivalent to $\mathcal{L}_{R_1^T, A, \mathcal{C}}$. $\square$

More intuitively, we can reason that because no intramolecular hybridizations are allowed by R_1^T, the initial complexes can aggregate only into tree-like structures. No matter how convoluted

the original complexes are, paths through the resulting tree-like structures are well modeled by context-free languages.

Our model of self-assembly of DNA into tree-like structures has strictly more computational power than the model of linear self-assembly. However, it is still a far cry from universal computation. It turns out that when we attempt to model intramolecular interactions, in the form of cooperative binding sites, a much more powerful model results. We consider a particular case in the following section.

3.2.5 Two Dimensional Self-assembly is Universal

To prove that two dimensional self-assembly can be universal, it suffices to demonstrate a restricted class which is universal. We review the class of structures introduced in Winfree (1996b), which are geometrically based on a lattice of double crossover (DX) units of DAO type Fu and Seeman (1993). It was shown in Winfree (1996b) that the self-assembly of DX units can directly mimic the operation of an arbitrary one dimensional cellular automata system. An example is shown in Figure 3.14, where a simple blocked cellular automaton rule (corresponding to the unrestricted grammar G_P of Section 3.2.1, but without the termination rules) is used to generate a Sierpinski triangle pattern.

The model of self-assembly used here follows Properties (1-4), (5b), and (6e), and it is motivated by additional physical concerns. As shown in Figure 3.14, the hybridization events may now involve *two* binding sites arranged as a *slot*. Geometry becomes important; only sticky ends which are close to each other and arranged properly may form a slot where binding can occur. Physically, one sticky end of an unattached DX unit would hybridize to one side of the slot, followed shortly by (the now intramolecular) hybridization of the DX unit's other sticky end to the slot's other binding site. For full computational generality, it is critical that a DX unit which matches one site in a slot, but not the other site, will *not* hybridize to the lattice. Under appropriate conditions, DX units which bind to only one site in a slot would soon dissociate, while fully matching DX units would bind nearly irreversibly. We therefore model slot-filling as a single permanent binary event involving two binding regions, and T is chosen so that single-site binding will not occur.

We emphasize that this form of DNA self-assembly has not yet been demonstrated experimentally, although we report some preliminary results in Section 4.1.

We must define new self-assembly rules: R_2^T allows hybridizations allowed by R_1^T, and additionally allows two-region slot-filling hybridizations between complexes containing the subgraphs shown in Figure 3.15, so long as the total number of basepair edges in B is at least T. This rule is meant to model local geometry in complexes; it will be a good model only for certain structures, including (we believe) the ones used in our construction.

The following can be proved by construction:

Theorem 3. (1) For all recursively enumerable languages $\mathcal{L}$, there exists a positive integer T, a codebook $\mathcal{C}$, and a set of duplexes and DX units A such that $\mathcal{L} = \mathcal{L}_{R_2^T, A, \mathcal{C}}$. (2) For all positive integers T, codebooks $\mathcal{C}$, and sets of duplexes and DX units A, $\mathcal{L}_{R_2^T, A, \mathcal{C}}$ is equivalent to a recursively enumerable language.

The proof of (1) is based on the constructions in Winfree (1996b). As cellular automata are capable of universal computation, for example by directly simulating Turing machines, we conclude that two dimensional self-assembly is universal. (2) follows because there is an algorithm for generating all the complexes in $\mathcal{L}_{R,A}$ so long as R is computable: keep trying new hybridizations of complexes known to be in the language, and remember the resulting complex. $\square$

Although universal, one dimensional cellular automata are not often a convenient model for computing functions of interest, although they are faster and more efficient than 1-tape Turing
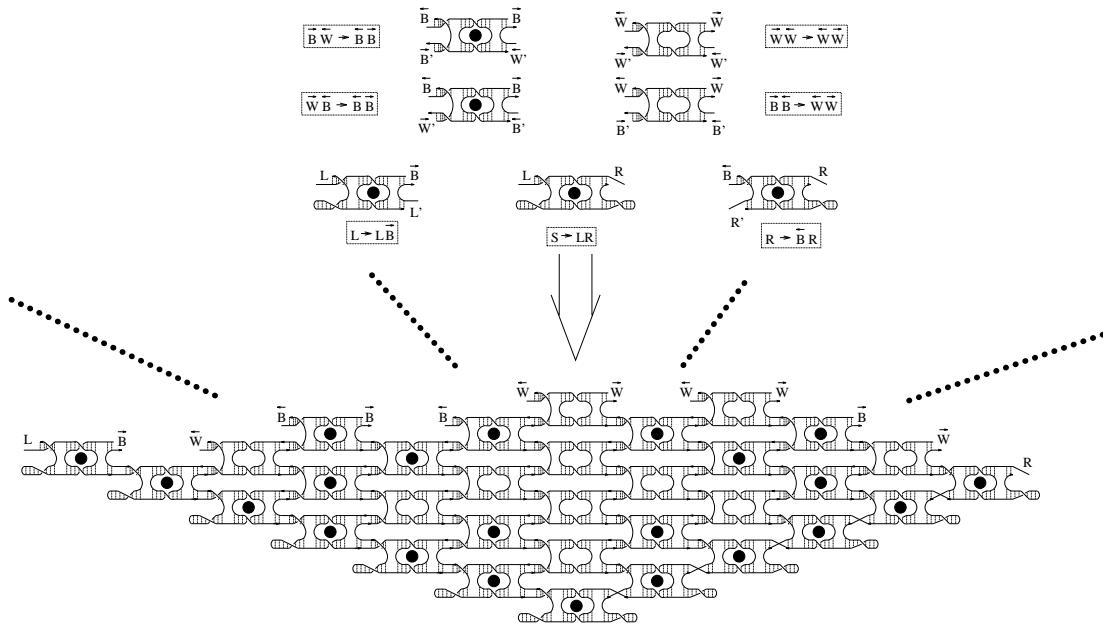
Figure 3.14: An algorithmic pattern in a self-assembled lattice. At the top, the seven initial DX units in A are shown (the black dot is a visual aid to identify "black" complexes), involving 22 oligonucleotides. The corresponding rewriting rules from G_P are presented in boxes. The units use 12 unique sticky end sequences, denoted by $\{L, R, \overleftarrow{B}, \overrightarrow{B}, \overleftarrow{W}, \overrightarrow{W}\}$ and their complements L' etc. The L and R sequences are both length T; the other sequences are length $T/2$. Upon self-assembly according to R_2^T, a V-shaped chain of the lower three units is formed due to hybridization of L and R, while the open slots in the initial chain are filled by the unique unit whose sticky ends match those on both sides of the slot. In this example, the process continues indefinitely. Each strand in $ligate(\hat{\mathcal{L}}_{R_2^T, A})$ represents one or two columns of Pascal's triangle mod 2.
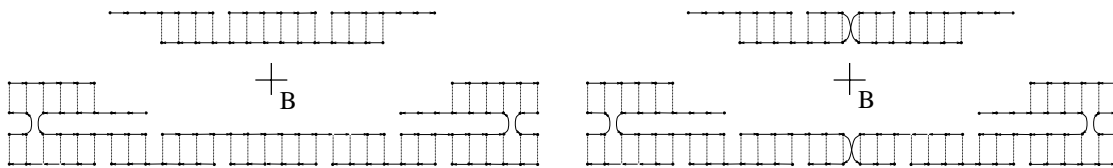
Figure 3.15: Two allowed slot-filling hybridizations in R_2^6. These graphs represent requires sub-graphs of the complexes C_1 and C_2 in $C_1 +_B C_2 = C_3$. Other positions of nicks are also allowed, as are other lengths of the duplex regions.

Machines, due to their parallelism.

3.2.6 Solving the Hamiltonian Path Problem

As a concrete example of using two dimensional self-assembly for computation, we will solve the same Hamiltonian Path Problem (HPP) used in Adleman (1994). Recall that the problem is to

find a path from node 1 to node N which visits every node in G exactly once. Our algorithms for solving HPP will be based on:

1. Generate all paths from node 1 to node N.

2. In each path, sort the vertices into increasing order.

3. For each path, check that the result is exactly "$1, 2, 3, \ldots, N$".

4. Output any path which passes the test, if one exists.

In a preparatory step, DNA sequences are designed for the given graph and synthesized. Steps 1-3 will occur as a single self-assembly step, while Step 4 consists of sequencing circular DNA of known length.

For the graph used in Adleman (1994) (shown in Figure 3.16a), $N = 7$ and we will require a total of 68 DX units of DAE type. Shown in Figure 3.16b, units 1 through 20 are responsible for Step 1 of the algorithm (the bottom layer in Figure 3.17a,b); these units are analogous to the oligos in Adleman's solution[29]. Units 19 through 61 are responsible for Step 2 of the algorithm; sorting is accomplished by the Odd-Even Transposition Sort (Knuth 1973). When the symbol ∞ has travelled all the way to the right, the sorting is complete and Step 3 is initiated, using units 62 through 68.

Each terminal complex either (a) encodes a valid Hamiltonian Path, in which case the complex is complete (Figure 3.17a), and ligation cyclizes the outer ring, but not the inner ring[30]; or (b) encodes an invalid path, in which case the terminal complex contains unfilled, open slots (Figure 3.17b) and will produce no cyclic strands when ligated[31]. Thus Step 4 can be achieved by separating cyclic from linear DNA strands (e.g. by 2D gel electrophoresis, by exonuclease digestion, or by affinity purification based on the $DONE$ sequence) followed by amplification and sequencing.

Let us briefly compare this molecular algorithm to the one used in Adleman (1994). To solve a graph with N nodes and E edges, Adleman used roughly $N + E$ oligos and N laboratory steps. We would use roughly $E^2/N + N^2 + N$ DX units (each requiring up to 5 strands) and a constant number of laboratory steps (synthesis, annealing, sequencing)[32].

Because two dimensional self-assembly can simulate arbitrary cellular automata, similar algorithms can be designed for any computational purpose. For example, an N-variable size s Circuit-SAT problem can be solved using roughly Ns DX units and a constant number of laboratory steps after synthesis.

---

[29]Adleman's oligos encoded individual edges in the graph, whereas ours encode pairs of edges. Also, knowing that a Hamiltonian path in this graph must visit exactly 7 nodes, our units are devised such that only odd-length paths can form completely.

[30]This can be ensured either by leaving an unmatched base on the sticky ends for interior units, or by phosphorylating only units which occur on the outer edge.

[31]Note that if a path visits a node twice, there will be a gap in the "Step 2" portion of the terminal complex; if a path fails to visit some node, there will be a gap in the "Step 3" portion of the terminal complex.

[32]How feasible these imagined laboratory steps would be is, of course, an open question. However, once the laboratory techniques have been debugged, conceivably our algorithm could be carried out in a single day's work – regardless of the size of the graph (volume permitting). A concern is that, more so than in Adleman's algorithm, the success of our algorithm is critically dependent upon ligation yields. For example, if ligation is 80% effective, then only $0.8^{30} = 0.1\%$ of the correct terminal complexes will be fully cyclized in our $N = 7$ graph. Also, since each path requires a DNA molecule roughly 100 times larger than the DNA used in Adleman's algorithm, greater reaction volumes will be necessary.

3.2.7 Three Dimensional Self-Assembly Augments Computational Power

A trivial corollary of the universality of two-dimensional self-assembly is that if three dimensional structures are allowed, self-assembly is still universal. It is of greater interest if we can exploit all three dimensions to allow for more efficient or more reliable computations. We propose a scheme
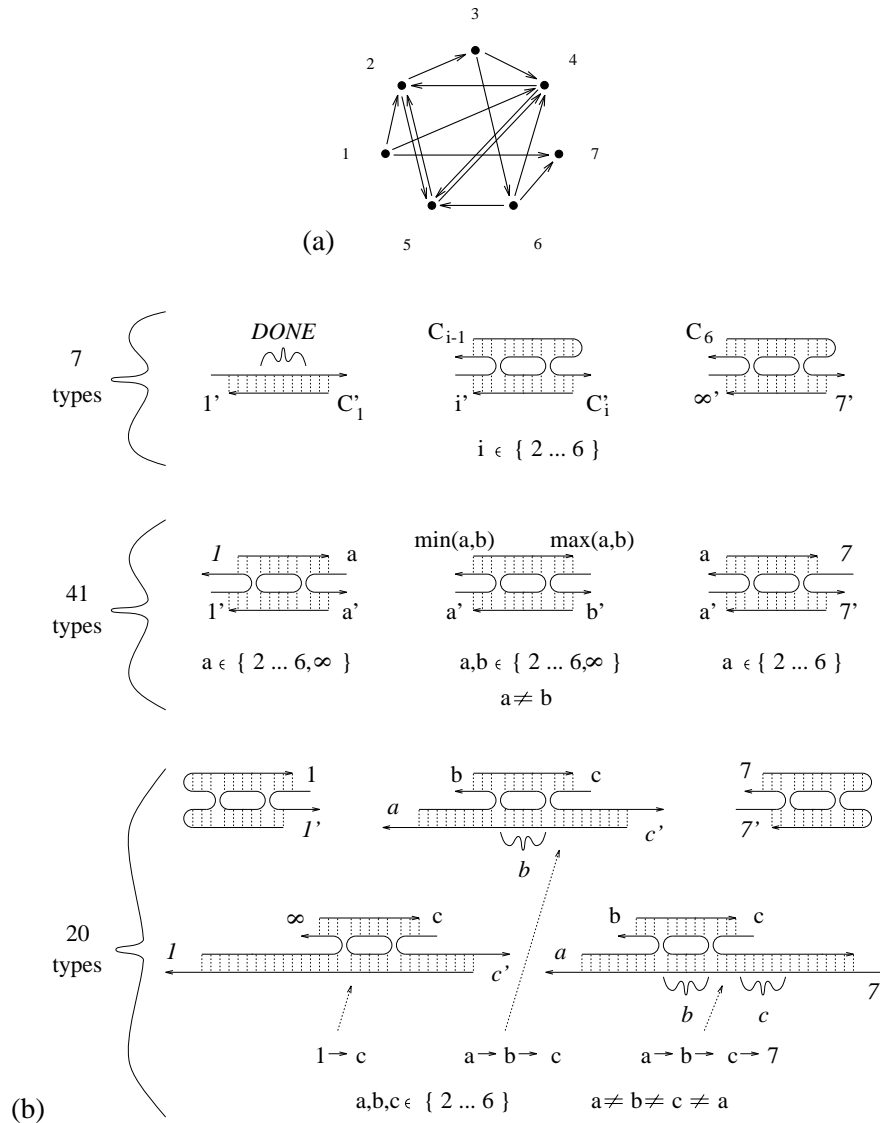
Figure 3.16: (a) The 7 node graph G. (b) Rule molecules (DAE type) for solving the Hamiltonian Path problem. Sticky ends of length $T/2$ are $\{1, 2, 3, 4, 5, 6, 7, \infty, C_1, C_2, C_3, C_4, C_5, C_6\}$ and their complements; sticky ends of length T are $\{1, 2, 3, 4, 5, 6, 7\}$ and their complements. $DONE$ is a special sequence which indicates completion of a lattice. The variables i, a, b, c are used to concisely represent a multiplicity of rule molecules; italicized variables indicate length T sticky ends. For example, in the lower set, 15 units are designed from the central schema, one for each pair of incident paths $a \rightarrow b, b \rightarrow c$.

(a)

(b)
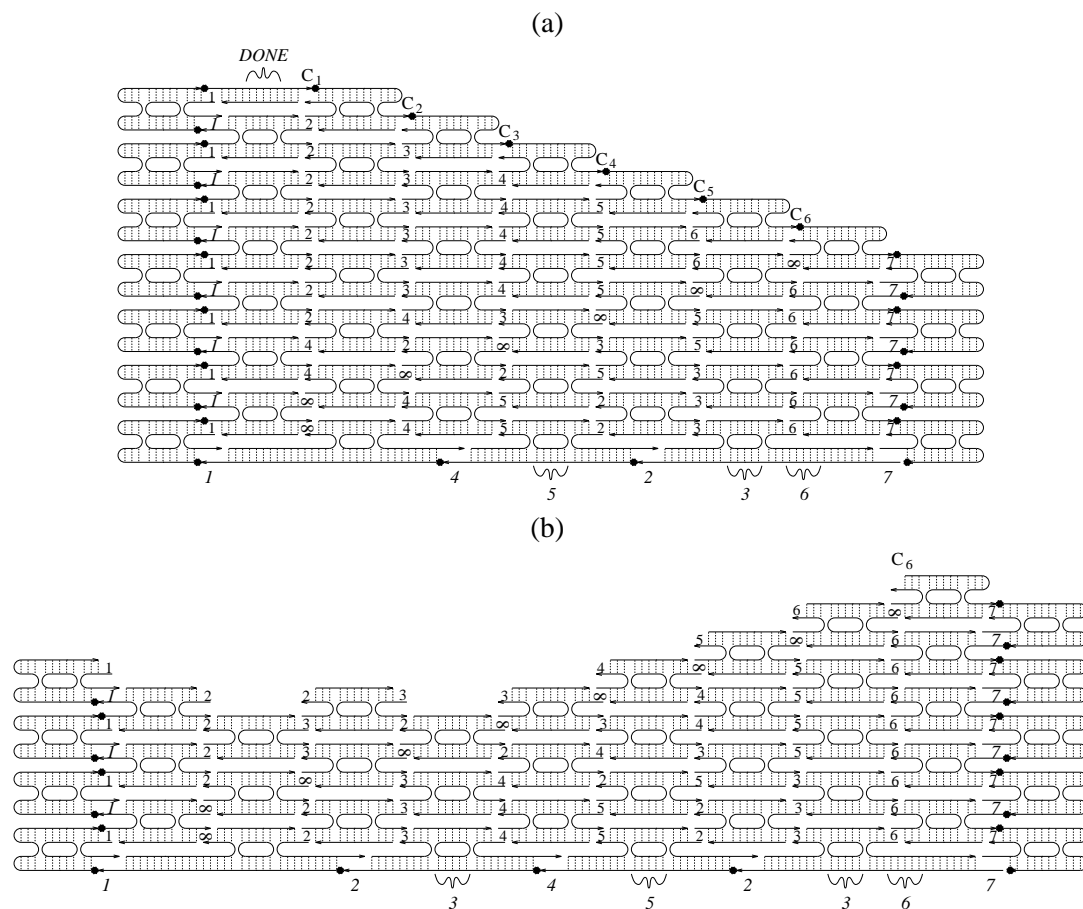
Figure 3.17: Terminal complexes after annealing. Black dots show nicks which will be ligated. (a) The lattice verifying the Hamiltonian path 1452367. (b) The lattice rejecting the invalid path 123452367.

to do exactly that, again for concreteness using DAO units as our basic building block. In this section, we will present some physical considerations, but we will not formally define R_8^T.

We begin by noting that the solid angle between two adjacent DAO units is determined by the length of the linker arm between them. For the planar lattice, we choose a length such that the angle is approximately 180°. Alternatively, we can choose lengths such that the angle is near 120°, the appropriate value for a "honeycomb lattice" as shown in Figure 3.18.

As in the case of the two dimensional lattice of DAO units, computation is brought about by judicious choice of the sticky end sequences on several DAO units. The three dimensional lattice thus formed is equivalent to the space-time history of a 2D blocked cellular automata.

The particular incarnation of three dimensional lattice chosen here is clearly not unique, and it is suggested more as a brain-teaser than as a serious proposal; other geometries are possible, perhaps having preferable practical characteristics.
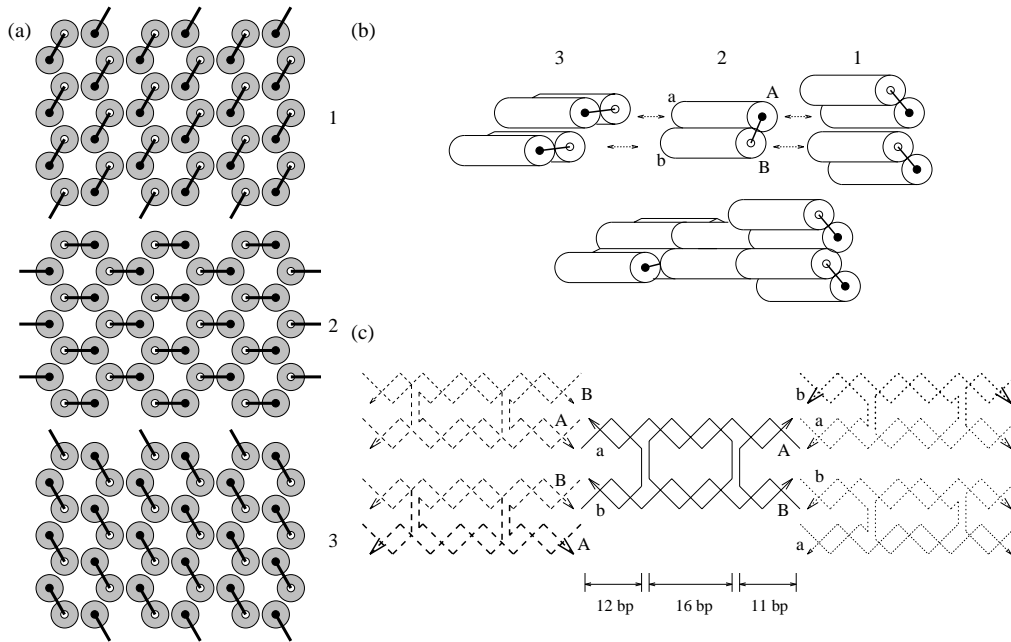
Figure 3.18: Plan for three dimensional lattice. (a) Three cross-sections through the final lattice, corresponding to the three sections indicated in (b). Circles represent cross-sections through a double helix of DNA; bars indicate which helices are part of the same DAO unit. (b) Relative angles of five DAO units are indicated. For perfect $120°$ angles, helical twist between 31.5 and $35.5°/$ bp is required. (c) Detail of the DAO units. A single DAO unit, with sticky end a complementary to A and b complementary to B, suffices to generate the entire lattice. For computations, the sticky ends are indexed s.t. a_i binds only A_i and b_i binds B_i.

3.2.8 Discussion

We have analyzed the computational power of three different regimes of self-assembly in our abstract model, and we have speculated on an extension into the self-assembly of a three dimensional lattice.

The essential construction in the linear case is due to Adleman (1994) who used it to construct paths through graphs. Boneh et al. (1996b) and Winfree (1996b) observed that linear self-assembly is capable of generating regular languages. Here, we state the result in the context of our formal model, and we show that linear self-assembly of duplexes is *limited* to regular languages. This point requires making the distinction between self-assembly processes with and without hairpins, as shown by the palindromes example. Linear self-assembly has been exploited in many laboratory experiments – both by molecular biologists and by people interested in molecular computation – and although its intricacies are not completely understood, there is a wide foundation of practical experience.

The self-assembly of branched junctions into dendrimer structures seems to be a relatively unexplored idea. For example, in Ma et al. (1986) it is observed that identical three-armed junctions with two complementary sticky ends can cyclize. If cyclization cannot be prevented, many context-free grammars would be impossible to implement by self-assembly. Another concern comes from geometry: if the desired tree-like structure contains too many branches, steric hin-

drance may prevent further associations from occurring. Thus it is not known to what extent the technique will be practical.

The self-assembly of DX units into a two-dimensional lattice is also an unconventional idea, yet to be demonstrated in the laboratory. Some first steps in this direction are reported in Section 4.1, where the slot-filling reaction is explored.

It is interesting to observe that the Chomsky Hierarchy of languages, developed originally for the study of human languages, also arises naturally in the study of self-assembling structures. The progression from regular to context-free to recursively enumerable languages can be seen to parallel both (a) the progression from linear to dendrimer to planar lattice structures, and (b) the progression from "rule molecules" with effectively one input and one output, to those with one input and two outputs, to those with two inputs and two outputs.

One should note that all the previous arguments ignored the kinetic framework implicit in the process of annealing that we originally consider. Specifically, we expect that longer complementary regions hybridize first. Annealing could be represented in our model as recursive computation of languages:

$$\mathcal{L}_{R_1,A,\mathcal{C}}^{anneal} = \mathcal{C}(denature(ligate(\hat{\mathcal{L}}_{R_1^1,\hat{\mathcal{L}}_{R_1^2,\hat{\mathcal{L}}_{R^3,\dots}}})))$$

The kinetic aspects of this model of linear self-assembly may themselves be exploitable for computation. Intermolecular interactions other than the ones considered here might also provide computational advantages. Issues of concentrations and finite supply of DNA must also go into any more practical analysis.

3.3 Simulation of Self-Assembly Thermodynamics and Kinetics

Abstract[33]   Winfree (1996b) proposed a Turing-universal model of DNA self-assembly. In this abstract model, DNA double-crossover molecules self-assemble to form an algorithmically-patterned two-dimensional lattice. Here, we develop a more realistic model based on the thermodynamics and kinetics of oligonucleotide hydridization. Using a computer simulation, we investigate what physical factors influence the error rates (i.e., when the more realistic model deviates from the ideal of the abstract model). We find, in agreement with rules of thumb for crystal growth, that the lowest error rates occur at the melting temperature when crystal growth is slowest, and that error rates can be made arbitrarily low by decreasing concentration and increasing binding strengths.

Early work in DNA computing (Adleman 1994; Lipton 1995; Boneh et al. 1996a; Ouyang et al. 1997) showed how computations can be accomplished by first creating a combinatorial library of DNA and then, through successive application of standard molecular biology techniques, filtering the library to identify the DNA representing the answer to the mathematical question. In these approaches, the problem to be solved determines the sequence of laboratory operations to be performed; the length of this sequence grows with problem size, intimidating many experimental researchers. Consequently, a few researchers have begun looking into chemical systems capable

---

[33]Results in this section also appear in Winfree (in press a).

of performing many logical steps in a single reaction, thus leading to paradigms for DNA computing where the problem to be solved is encoded strictly in DNA sequence; a fixed sequence of laboratory operations is performed to determine the answer to the posed question. Promising approaches include techniques based on PCR-like reactions (Hagiya et al. in press; Sakamoto et al. in press ; Hartemink and Gifford in press; Winfree in press b) and techniques based on DNA self-assembly (Winfree 1996b; Winfree et al. in press; Jonoska et al. in press). Although there has been experimental work exploring all these models, typically only a few logical operations have been demonstrated. It is at this point unclear how well any of the techniques can be scaled up. Short of full experimental demonstration, realistic simulations of the chemical kinetics and thermodynamics can shed light on what can be expected of these systems, and can point to parameter regimes where the experiments are most likely to succeed. This paper presents a preliminary analysis of the self-assembly model of Winfree (1996b).

To motivate the self-assembly model, we consider the physical process of crystallization. During crystal growth, monomer units are added one-by-one at well-defined sites on the surface of the crystal. There may be more than one type of monomer, in which case there may be several different types of binding site, each with affinity for a different monomer; typically a periodic arrangement of units results. The question of whether periodic lattices will *necessarily* result has been studied in mathematics in the context of two-dimensional tilings (Grünbaum and Shephard 1986). A set of geometrical shapes (the *tiles*) are said to tile the plane if the tiles can be arranged, non-overlapping, such that every point in the plane is covered. A surprising result in the theory of tilings is that there exist sets of tiles which admit *only* aperiodic tilings (Berger 1966; Robinson 1971), the most elegant being the rhombs of Penrose (1978). The variety of aperiodic patterns is limitless: using square tiles with modified edges, the time-space history of any Turing Machine can be reproduced by the tiling pattern[34] (Wang 1963; Robinson 1971). Is it possible to translate these results back to a physical system, to produce aperiodic crystals, or even crystals which "compute"? Already, there is an extensive literature on "quasicrystals" (Steinhardt and Ostlund 1987), materials which exhibit "prohibited" 5-fold symmetry and which are thought to be related to the aperiodic Penrose tiles. The purpose of this paper is to examine the suggestion in Winfree (1996b) that DNA double-crossover molecules can be used to make programmable "molecular Wang tiles" that will self-assemble into a 2D sheet to simulate any chosen cellular automaton. It has already been shown experimentally that double-crossover molecules can be designed to assemble into a periodic 2D sheet (Winfree et al. 1998) and that a single logical step can proceed in a model system. In this paper we argue that it is physically plausible to perform Turing-universal computation by crystallization.

3.3.1 An Abstract Model of 2D Self-Assembly

The results in the theory of tilings are entirely existential, saying nothing about *how* a correct tiling is to be found. What is missing is a mechanism for producing tilings. In this section we describe the relation of computation and self-assembly by presenting an abstract model of two-dimensional (2D) self-assembly, which we call the Tile Assembly Model. The fundamental units in this model are unit square *tiles* (also called *monomers*) with labelled edges. We have an unlimited supply of tiles of each type. *Aggregates* are formed by placing new tiles next to and aligned with existing ones such that sufficiently many of their edges have matching labels. Tiles cannot be rotated or reflected. To define the model completely, we must be precise about when "sufficiently many"

---

[34]Even more is possible: there exist tile sets which produce *non-recursive* patterns (Hanf 1974; Myers 1974)! However, it is unlikely that any physical process could give rise to non-recursive patterns, in any computable amount of time. All models discussed in this paper are strictly computable.

edges match. Each edge label σ_i has an associated *strength* g_i, which must be a non-negative integer. At "temperature" $\mathcal{T}$, an aggregate of tiles can grow by addition of a monomer whenever the summed strength of matching edges exceeds $\mathcal{T}$ (mismatched labels neither contribute nor interfere) – these are called *stable* additions. We say that a set of tiles P *produces* aggregate A from *seed* tile T if A can be obtained from the single tile T by a sequence of zero or more stable additions of monomers; in which case, we also say simply that P produces A (if there is no need to specify the seed tile).

To illustrate this model, consider the 7 tiles shown in Figure 3.19d. The four tiles on the left are called the *rule tiles* because they encode addition mod 2; the three tiles on the right are the *boundary tiles*; the one with two strength-2 edges is the *corner tile*. There are 4 edge labels, of strengths 0, 1, 1, and 2. At temperature $\mathcal{T} = 0$, every possible monomer addition is stable, and thus random aggregates are produced. At temperature $\mathcal{T} = 1$, at least one edge must match for an addition to be stable, but now the arrangement of tiles within an aggregate depends upon the sequence of additions. At temperature $\mathcal{T} = 2$, there is a unique choice for the tile in each position relative to the corner tile, independent of the sequence of events. Under these conditions, this set of tiles produces the Sierpinski Triangle by computing Pascal's Triangle mod 2. At temperature $\mathcal{T} = 3$, no aggregates are produced because no monomer addition to another monomer is stable.
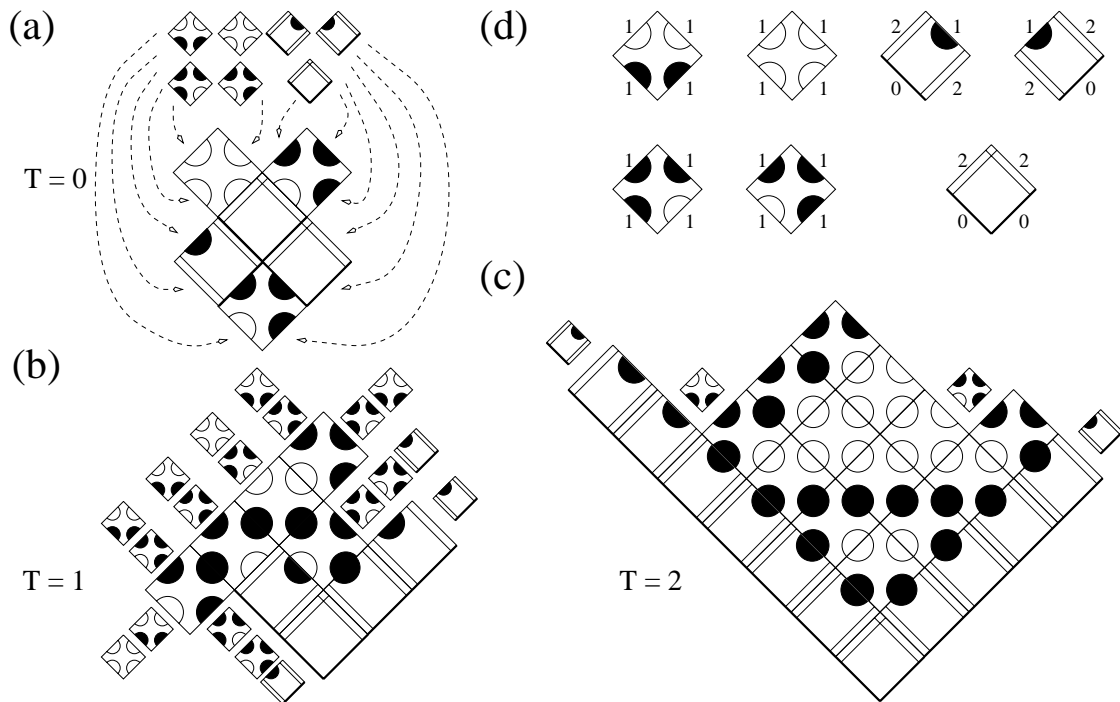
Figure 3.19: The Sierpinski Tile set is shown in (d). The strengths of edges are marked, and the edge labels are denoted graphically. In (a) - (c), small tiles are used to indicate possible stable additions to the aggregate. (a) When $\mathcal{T} = 0$, any tile addition is stable, and a random aggregate results. (b) When $\mathcal{T} = 1$, typically several stable possibilities at each site; again, a random aggregate results. (c) When $\mathcal{T} = 2$, there is a unique possibility at each site, resulting in unique pattern formation.

Whereas it is impossible to uniquely produce non-trivial aggregates when $\mathcal{T} = 0$, an arbitrary

shape can be produced at $\mathcal{T} = 1$ by assigning a unique tile to each position and giving each edge a unique label. However, this requires the use of many tiles. At $\mathcal{T} = 2$ we can produce interesting patterns with few tiles.

A hint of the computational power of the Tile Assembly Model when $\mathcal{T} = 2$ is provided by a simulation of cellular automata[35]. The proof we develop below demonstrates two important points. First, even though tile addition is stochastic, a unique pattern is produced regardless of the order of events, because only stable tile additions are made. Second, the arrangement of tile types on the 1D growth front of the aggregate can represent information (much like how the arrangement of 0's and 1's on a 1D tape represents information for a Turing Machine), and stable tile additions can modify that information by specified rewrite rules, resulting in fully general computation.

Our simulation is based on one-dimensional blocked cellular automata (BCA)[36], a variety of cellular automaton (CA). The example of Pascal's Triangle mod 2 (Gardner 1966) has been studied as a cellular automaton by Wolfram (1984). It is known that BCA and CA are Turing-universal models, and simple simulations of Turing machines have been demonstrated (Smith 1971; Biafore preprint). We begin by defining BCA.

Definition: A k-symbol BCA is defined (using the integers $\{1, 2, \ldots, k\} = \mathcal{Z}_k$) by a rule table

$$R = \{(l_i, r_i) \to (l'_i, r'_i)\} \subset (\mathcal{Z}_k \times \mathcal{Z}_k \to \mathcal{Z}_k \times \mathcal{Z}_k).$$

If R is a function, then the BCA is termed deterministic. The *state* $\mathbf{c}$ of the BCA assigns a symbol to every location on an infinite linear array of *cells*. At each time step every cell in $\mathbf{c}$ is rewritten to produce $\mathbf{c}^{t+1}$; thus we use $c^t(x)$ to denote the symbols written in cell x after t steps. The BCA uses R to re-write pairs of cells in $\mathbf{c}$, alternating between even and odd alignments of the pairing: for even t and even x, and for odd t and odd x,

$$\Big((c^t(x), c^t(x + 1)) \to (c^{t+1}(x), c^{t+1}(x + 1)) \Big) \in R.$$

An input to a BCA computation is a state $\mathbf{c}^0$ with a finite number of non-zero cells. For convenience and without loss of generality, we will confine our attention to n-bit binary inputs $\mathbf{b}$, and write $\mathbf{c}^0 = \mathbf{b}$ to refer to an input where $c^0(i) = b_i$ for $1 \le i \le n$ and $c^0(i) = 0$ otherwise.

The computation of the BCA defines $c^t(x)$ over the half-plane $t \ge 0$. We will show how to construct a set of tiles P such that in all aggregates produced from the seed tile T_0, if there is a tile at position (i, j) with respect to the seed tile, then the tile has edges encoding $c^{i+j}(i - j)$ and $c^{i+j}(i - j + 1)$. Thus the time-history of the BCA computation is reproduced exactly in the self-assembled tile aggregate.

First we show, for any n-bit BCA input $\mathbf{b}$, how to generate the set of $n + 3$ *input tiles* $I(\mathbf{b})$. Figure 3.20a shows the construction. Because the only edge matches possible with these tiles are strength 2, at $\mathcal{T} = 2$ all produced aggregates are essentially as shown, with variable length regions encoding "zero" on either side. The tile whose top edges encode bits b_1 and b_2 is referred to as the seed tile T_0 and is used as the reference for indexing tiles by location. The bottom of each

---

[35]This result, presented in less detail in Winfree (1996b), translates Wang's simulation of Turing Machine execution by the Tiling Problem (Wang 1963) into the Tile Assembly Model given here. The Tiling Problem can be viewed as asking for the ground state of an N-state Ising model, which can be seen as a question of equilibrium thermodynamics in the limit as $T \to 0$. Not only can Ising models be produced which are Turing-universal because the ground state reproduces the space-time history of any chosen Turing Machine, but the proof that tiles sets can be found which tile the plane non-recursively shows in fact that the ground state of an Ising model can be non-recursive. Thus it is essential to study a kinetic, rather than thermodynamic, model.

[36]BCA (Wolfram 1994) are also known as *partitioning* CA (Margolus 1984) and as 2-body CA or particle machines. They generalize the lattice gas model (Hardy et al. 1976), and are commonly studied in two dimensions.

input aggregate contains only strength-0 edges, so no further additions can occur there. The top of each input aggregate contains exclusively strength-1 edges, arranged in a zig-zag forming series of binding sites, called *slots*, where a new tile could make contact with *two* strength-1 edges. For aggegates containing the seed tile T_0, these edges encode the input c_0 and the pairing of cells.
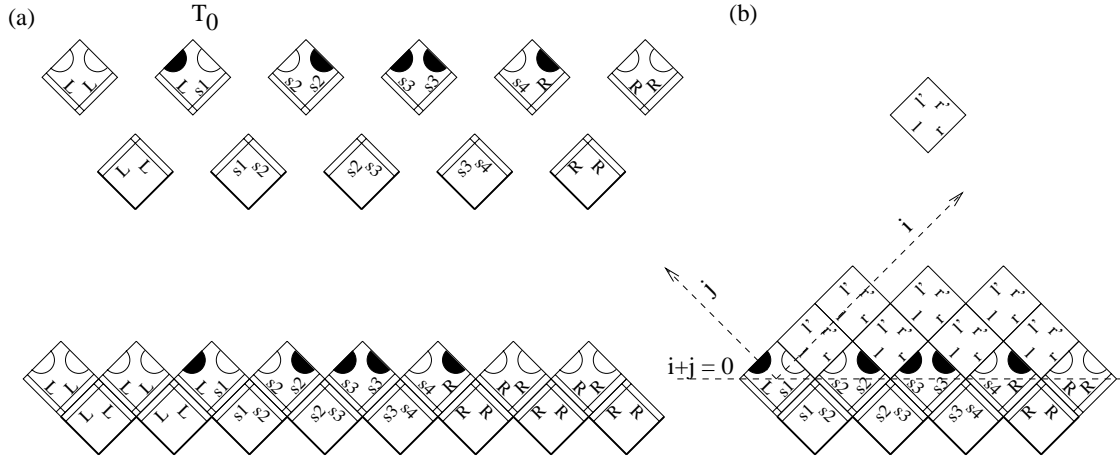

Figure 3.20: Using the Tile Assembly Model to simulating a BCA computing from a binary input. (a) Input tiles $I(\mathbf{b})$ for $\mathbf{b} = 10011101$, and an aggregate they produce at $\mathcal{T} = 2$. Here we use conventions similiar to Figure 3.19 to indicate the strength of edges: thick edges are strength-0, doubled edges are strength-2, and all other edges are strength-1. (b) Schematic showing a rule tile generated from the BCA rule $(l, r) \rightarrow (l', r')$, and an aggregate produced by the rule tiles and input tiles. Note the dotted lines indicating the default coordinate system with origin at the seed tile T_0. In this schematic, the edge labels for the rule tiles are not identified.

Next, for BCA rules R we generate a set P_R of k^2 tiles as shown in Figure 3.20b, using one tile for each rule $(l, r) \rightarrow (l', r')$. All of these *rule tiles* have exclusively strength-1 edges, so at $\mathcal{T} = 2$ they cannot form aggregates with themselves; they must be seeded by the input tiles. Thus, when the tile sets P_R and $I(\mathbf{b})$ are mixed, rule tiles can sit down in the slots presented by the input aggregates iff both of the presented edges match. Consider an aggregate in which: (1) only rule tiles are present above $i + j = 0$, and (2) every rule tile has both of its lower edges correctly matched. It follows directly from the definitions that the edges presented by the tile at (i, j) has edges encoding $c^{i+j}(i - j)$ and $c^{i+j}(i - j + 1)$ because this is true of the input tiles, and every rule tile respects the update rule for the BCA. What remains to be shown is that (1) and (2) hold for every aggregate produced at $\mathcal{T} = 2$. This is done by induction on N, the number of rule tiles in an aggregate. For convenience, we refer to an aggregate containing exactly N rule tiles as an N-aggregate.

Base case: (1) and (2) hold for any 0-aggregate.

Induction: Assume (1) and (2) hold for all N-aggregates. Note that (1) and (2) together imply that above $i + j = 0$, the exposed edges of the aggregate are all *upper* edges. Any $(N + 1)$-aggregate must be produced from an N-aggregate by a sequence of stable additions of input tiles followed by a stable addition of a rule tile. (1) holds for the new aggregate because all exposed edges above $i + j = 0$ are upper edges labelled from $\mathcal{Z}_k$, while all lower edges of input tiles are labelled from $\{L, R, s_1, \ldots, s_n\}$. (2) holds for the new aggregate because a rule tile must match

two edges to be added, and only upper edges are presented, so the rule tile's two lower edges must match. □

Thus, we have proven:

Theorem: Let R be a BCA, and let $c(t, x)$ be the value of cell x at time t for a computation on input **b**. If an aggregate produced from seed T_0 by the tile set $P = P_R \cup I(\mathbf{b})$ has a tile in position (i, j), then the tile's upper edges encode $\dot{c}^{+j}(i - j)$ and $c^{i+j}(i - j + 1)$.

In other words, the Tile Assembly Model uses asynchronous and self-timed updates to simulate any deterministic one-dimensional BCA. Similar arguments can be used to show that the Tile Assembly Model can simulate any *non-deterministic* one-dimensional BCA, in the sense that every possible aggregate produced according to the Tile Assembly Model will represent a possible history of execution of the non-deterministic BCA. In this case, R will contain rules with identical left-hand sides, and consequently in some slots multiple rule tiles will match both exposed edges; thus a non-deterministic choice must be made. Alternatively, a non-deterministic set of input tiles may be used to generate a combinatorial set of possible input strings, followed by deterministic evaluation of each input. The potential for non-determinism is important for using self-assembly to solve combinatorial search problems in the spirit of Adleman (1994).

3.3.2 Implementation by Self-Assembly of DNA

We follow Winfree (1996b) in developing a molecular implementation of the Tile Assembly Model: each tile is represented by a DNA double-crossover (DX) molecule (Fu and Seeman 1993) with four sticky ends whose sequences represent the edge labels. We would like these molecular "tiles" to self-assemble into a two-dimensional sheet according to the rules of the Tile Assembly Model (see Figure 3.21). Thus, we need to show:

1. Double-crossover molecules can designed to self-assemble into two-dimensional crystal lattices – in preference over, for example, random tangled nets, tubes, or other structures. This has in fact now been demonstrated in an experimental system (Winfree et al. 1998).

2. The strengths of edge labels in the model can be implemented by designing the sticky end sequences with specific energetics of hybridization. The DNA hybridization strengths depend primarily on the number of base pairs, with adjustments for their particular sequence, the buffer conditions, and temperature. Thus, for example, longer sticky ends can be used to represent edge labels with greater strength.

3. The binding of DX molecules into slots, where two sticky end sequences must both hybridize, is *cooperative* – thus, strengths "add". We will argue below that this is *a priori* likely; furthermore, suggestive experimental evidence has been presented in Winfree et al. (in press).

4. There is a physical parameter analogous to $\mathcal{T}$ which determines the strength required for association of molecular tiles. This parameter can be, for example, the temperature T. DNA sticky ends bind more strongly at low temperatures, and conversely, at higher temperatures more sticky-end interactions will be necessary for stable addition.

5. All these considerations can come together to produce molecular self-assembly in accordance with the Tile Assembly Model.
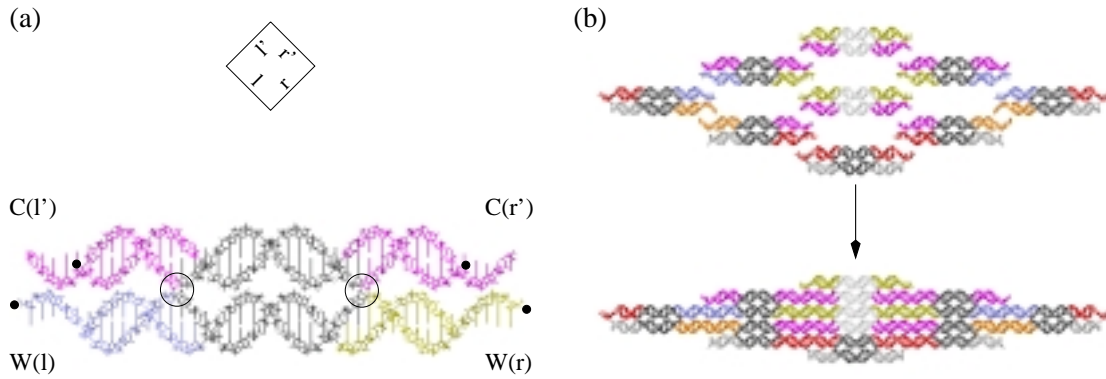
(a)

C(l') C(r')

W(l) W(r)

(b)

Figure 3.21: The DNA representation of Wang tiles. (a) A molecular Wang tile (double crossover molecule) representing the rule $(l, r) \rightarrow (l', r')$. The molecule consists of an interior structural region and four double-stranded arms, each terminated by a single-stranded sticky end. Edge labels are implemented using unique sticky-end sequences. Note that sticky-ends for the lower edges use Watson-sense sequences for each label, while the upper edges use the complementary Crick-sense sequences. This ensures the proper relative orientation of tiles. As shown, the same molecule represents both a Wang tile and its reflection about the vertical axis; however, using four encodings for each label ($W_{left}, W_{right}, C_{left}, C_{right}$) eliminates reflection-sense binding. In the double crossover molecule, the crossover points are circled, and dots are placed at the 5 ends of each strand. Color is used to indicate the edge label being represented, and not the identity of strands (each strand is multi-colored). (b) The self-assembly of 9 molecular Wang tiles, of 5 distinct types. These correspond to the 9 tiles at the bottom of Figure 3.19c. Note that the corner and boundary molecules have hairpin sequences, and thus no sticky ends, on certain of their lower arms; this implements a tile with strength-0 labels on its lower edges. Also note that on the corner and boundary molecules, the red and orange sticky ends are sufficiently longer than the sticky ends on the rule molecules to implement a strength-2 interaction.

Our approach for arguing these points is based on the study of the thermodynamics and kinetics of DNA oligonucleotide hybridization (Wetmur 1991). We review here the elements of this theory that are needed for our discussion.

Let ssDNA$_1$ and ssDNA$_2$ be two Watson-Crick complementary oligonucleotides, and let ds-DNA be the double-stranded helical complex that results upon their hybridization. The reaction can be modelled as a two-state first-order system:

$$\text{ssDNA}_1 + \text{ssDNA}_2 \; \underset{k_r}{\overset{k_f}{\rightleftharpoons}} \; \text{dsDNA}.$$
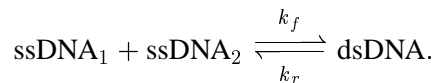
We can write a differential equation for the rates of change of the concentration of each species. The units for k_r are 1/sec, so $k_r\,[\text{dsDNA}]$ gives the rate in M/sec of dissociation of the double helix; the units for k_f are 1/M/sec, so $k_f\,[\text{ssDNA}_1][\text{ssDNA}_2]$ gives the rate in M/sec of hybridization to form new double helical molecules. Altogether, we have:

$$-[\dot{\text{dsDNA}}] = [\dot{\text{ssDNA}}_1] = [\dot{\text{ssDNA}}_2] = k_r\,[\text{dsDNA}] - k_f\,[\text{ssDNA}_1][\text{ssDNA}_2]$$

The rate constants k_f and k_r can be estimated from the DNA sequence and the temperature T (in K), assuming the reaction is taking place in a standard buffer. For very short oligonucleotides, the forward reaction has a diffusion-controlled rate-determining step (Quartin and Wetmur 1989) approximately independent of oligo length and sequence, so:

$$k_f = A_f \ e^{-E_f/RT} \approx 6 \times 10^5 \ /\text{M/sec},$$

where $A_f = 5 \times 10^8$ /M/sec and $E_f = 4\,\text{kcal/mol}$ is the activation energy for the reaction[37]. The reverse rate, on the other hand, is very sensitive to oligo length and sequence:

$$k_r = k_f e^{\Delta G_s^\circ/RT},$$

where $R = 2\,\text{cal/mol/K}$ and $\Delta G_s^\circ < 0$ is the free energy released as heat by a single hybridization event[38]. The standard free energy ΔG_s° can be calculated from the standard enthalpy ΔH_s° and the standard entropy ΔS_s° of the reaction: $\Delta G_s^\circ = \Delta H_s^\circ - T\Delta S_s^\circ$. For reactions taking place in commonly used buffers, the standard enthalpy and entropy can be reliably estimated from the sequence according to a nearest-neighbor model (SantaLucia et al. 1996); however, for the purposes of this discussion, we can use the coarser approximation for length-s oligonucleotides[39]: $\Delta H_s^\circ \approx -8s\,\text{kcal/mol}$ and $\Delta S_s^\circ \approx -22s - 6\,\text{cal/mol/K}$. Thus we can predict both k_f and k_r for the hybridization of complementary oligonucleotides. This allows us to predict the equilibrium concentrations of each species via the equilibrium constant

$$K \doteq \frac{[\text{dsDNA}]}{[\text{ssDNA}_1][\text{ssDNA}_2]} = \frac{k_f}{k_r} = e^{-\Delta G_s^\circ/RT}.$$

We will use our understanding of oligonucleotide hybridization kinetics and thermodynamics to build a plausible model for the self-assembly of DX molecules via the hybridization of their sticky ends.

3.3.3 A Kinetic Model of DNA Self-Assembly

The self-assembly of two-dimensional lattices from a heterogeneous mix of N DX molecules is a far more complicated system than the hybridization of two oligonucleotides. Rather than having just three species to consider (ssDNA$_1$, ssDNA$_2$, and dsDNA), we now have an infinite number of species (all possible aggregates). For each aggregate of n tiles with m available sites, there are Nm association reactions and n dissociation reactions. Note that at every available site, there is an association reaction for every possible monomer, regardless of whether the monomer is the "correct" one or not; to understand when correct behavior can be expected, we must look closely at the kinetics of all the reactions. The model we develop here can be seen as an extention of Erickson (1980), which considers the self-assembly of an isotropic two-dimensional lattice consisting of a single unit type. To model the kinetics of self-assembly, we make several simplifying assumptions:

1. Monomer concentrations will be held constant. Further, all monomer types will be held at

---

[37]We will ignore the activation energy in what follows, because we will see that the value of k_f has no effect on the behavior of the system except to set the scale of the time axis.

[38]The more negative ΔG_s° is, the more heat is released upon association and the more favorable the reaction is. Another way of looking at it is that if ΔG_s° is very negative, a lot of heat must simultaneously converge upon a single double helical DNA molecule in order to cause dissociation, and thus dissociation is rare. Also note that here, as elsewhere, $e^{\Delta G_s^\circ/RT}$ has an "invisible" unit of M, so that k_r is in units of 1/sec.

[39]The empirical value $\Delta S_{init} = -6\,\text{cal/mol/K}$ can be considered the entropic cost of aligning the two strands to have the same orientation, and is called the initiation entropy.

the same concentration. Primarily we make this assumption because the analysis is easier. Later we show how the results found with the assumption can be used to understand the more general case when the assumption is not true[40].

2. Aggregates do not interact with each other; thus the only reactions to model are the addition of a monomer to an aggregate, and the dissociation of a monomer from an aggregate. Potential drawbacks of this assumption will be discussed at the very end.

3. As in the hybridization of oligonucleotides, we assume that the forward rate constants for all monomers are identical. In particular, the forward rate constants for correct and incorrect additions are identical.

4. As in the hybridization of oligonucleotides, we assume that the reverse rate depends exponentially on the number of base-pair bonds which must be broken, and that mismatched sticky ends make no base-pair bonds. This amounts to assuming that binding on multiple edges is cooperative and that mismatched sticky ends do not affect the dissociation rate in any way.

The model is governed by two free parameters, both of which are dimensionless free energies: $G_{mc} > 0$ measures the entropic cost of fixing the location of a monomer unit (and thus is dependent upon monomer concentration), and $G_{se} > 0$ measures the free energy cost of breaking a single sticky-end bond; both are expressed with respect to the thermal energy RT. A third parameter, the forward rate constant k_f, is immaterial to the behavior of the system; it sets the units for the time axis. The behavior of the system can be understood independently of the exact correspondence of these abstract parameters to more realistic physical parameters; however, we sketch the correspondence below.

For convenience, we lump location, orientation, and other entropic factors together into an "effective concentration" of monomers, $[\hat{D}X]$. In these units, $[\hat{D}X] = [DX]/20$, $\hat{k}_f = 20k_f$, and the initiation entropy of $\Delta S_{init} = -6\,\text{cal/mol/K} = -R\ln 20$ disappears from the equations. Now we write the concentration of each monomer as $[\hat{D}X] = e^{-G_{mc}}$. Thus the rate of associations of a particular monomer type at a particular site on a particular aggregate is

$$r_f = k_f[DX] = \hat{k}_f e^{-G_{mc}},$$

measured in 1/sec. To determine the dissociation rate of a unit bound by b sticky-end bonds, each of length s, we will use our assumption of cooperativity to justify using the free energy of a single length-$b \cdot s$ oligonucleotide, $\Delta G^{\circ}_{b \cdot s}$. To write the dissociation rate in terms of G_{se}, we have:

$$r_{r,b} = k_f e^{\Delta G^{\circ}_{b \cdot s}/RT} = \hat{k}_f e^{-bG_{se}},$$

also measured in 1/sec. Using the values for ΔH°_s and ΔS°_s determined for oligonucleotide hybridization, sticky ends of length s would correspond to $G_{se} = (\frac{4000\,\text{K}}{T} - 11)s$. If strength-1 edge labels are encoded with sticky ends of length s ($b = 1$), then strength-2 edge labels will be encoded with sticky ends of length $2s$ ($b = 2$). If b is the sum of the strength of all a tile's matching edges, then the tile's dissociation rate will be $r_{r,b}$, and we will call b the number of (sticky-end) *bonds*.

The various reactions possible in this model, which we call the Kinetic Assembly Model, are illustrated for the Sierpinski Tiles in Figure 3.22.

---

[40]There is some intrinsic interest in the case where the assumption is true; for example, biological self-assembly often occurs in the context where genetic circuitry controls the concentration of the monomers via a feedback loop.
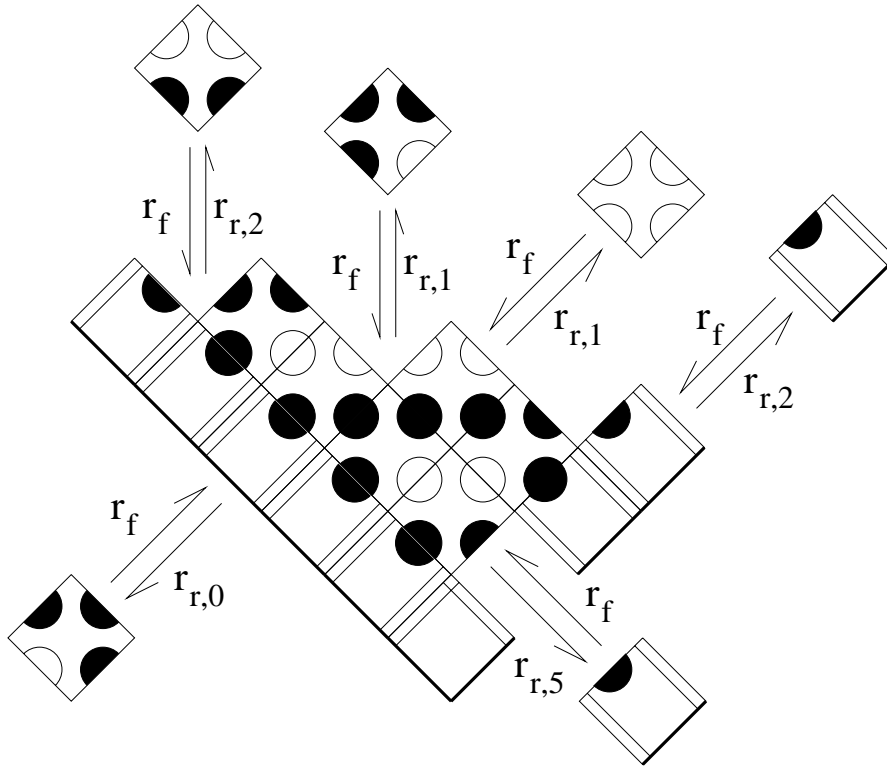
Figure 3.22: The rates of reactions for various tile association and dissociation steps in the Kinetic Assembly Model. Note that all on-rates are identical, and that off-rates depend only upon the total strength of correct edge matches. Mismatched edges and empty neighbors are treated identically.

We now wish to understand the behavior of the Kinetic Assembly Model as a function of it two free parameters, G_{mc} (controlled by monomer concentration) and G_{se} (controlled by temperature and by sticky-end length). Our naive prediction is that the ratio $\frac{G_{mc}}{G_{se}}$ plays the role of $\mathcal{T}$ in the Tile Assembly Model. If for small $0 < \epsilon < 1$

$$\mathcal{T} \doteq \frac{G_{mc}}{G_{se}} = b - \epsilon,$$

then for a tile with b matches at a site,

$$\frac{r_f}{r_{r,b}} = e^{bG_{se} - G_{mc}} = e^{\epsilon G_{se}} > 1,$$

and the site will tend to be filled. But a tile with $b - 1$ matches will have

$$\frac{r_f}{r_{r,b}} = e^{-(1-\epsilon)G_{se}} \ll 1,$$

and the tile will tend to dissociate. Because at equilibrium for the local site, the correct tile is preferred over incorrect tiles by a factor of $e^{G_{se}}$, we expect that for large G_{se}, the Kinetic Assembly Model will with high likeliness produce aggregates produced by the Tile Assembly

Model. To confirm this expectation and delineate when it applies, we will have to understand when local equilibrium is achieved, when the kinetics works in our favor, and when it works against us.

We begin our detailed analysis by simulating the behavior of the Kinetic Assembly Model. Because there are an infinite number of possible aggregate types, we cannot simply integrate the rate equations to determine the time evolution of the concentration of each aggregate type. However, since aggregates do not interact with each other, we can develop our simulation from the perspective of an individual aggregate, starting with a chosen seed unit. Reaction rates now become probability rates for a Poisson process: the association or dissociation of a monomer from the current aggregate. In such a simulation, the probability of observing a particular aggregate at simulated time t corresponds to the fractional concentration of that aggregate at time t according to the full model.

The simulation proceeds as follows: A 2D array is used to store the arrangement of tiles in the current aggregate. Initially the array contains all zeros to indicate empty sites, except for the origin, which contains the seed tile. To determine the next event, the rates of all possible reactions must be known. All m empty sites adjacent to the aggregate are counted; the net on rate is

$$k_{on} = m \hat{k}_f e^{-G_{mc}}.$$

For all occupied sites (i, j) within the aggregate (except for the seed tile at the origin), the tile types of its neighbors are noted and the total strength b_{ij} of all matching labels is calculated; the net off rate is $k_{off} = \sum_b k_{off,b}$ where

$$k_{off,b} = \sum_{ij \text{ s.t. } b_{ij}=b} \hat{k}_f e^{-b_{ij} G_{se}}.$$

Thus the net rate for events of any kind is $k_{any} = k_{on} + k_{off}$, and the time until the next event occurs, Δt, is chosen according to the Boltzman distribution $\Pr(\Delta t) = k_{any} e^{k_{any} \Delta t}$. Now, given that an event has occurred, the probability that it is an on-event is k_{on}/k_{any}, in which case all sites and all tile types are equally likely to be chosen; otherwise a dissociation has occurred, and the probability that some site with b bonds dissociates is $k_{off,b}/k_{off}$, and again all such sites are equally likely. Once the event is chosen and the array is updated, all rates must be recalculated to determine the next event[41].

3.3.4   Simulation Results

This section discusses simulations of the self-assembly of the Sierpinski Tiles using the Kinetic Assembly Model. An example run is shown in Figure 3.23. Several features of this simulation run warrant comment.

Shape: The growth front does not advance synchronously, but rather performs a biased random walk, with the following restriction: because stable addition occurs only at concave corner sites (slots) on the growth front, no sites can be more than one step ahead of or behind its neighbors. The growth front is concave on average: the boundary tiles grow fastest because their growth site is always available, while internal regions on the growth front grow slower because stable addition can occur at only a fraction of sites at any given time.

---

[41]The actual computer code is optimized to remove redundant calculations, of course!

Errors: For the most part, the Sierpinski Triangle is accurately reproduced. However, incorrect tiles do appear. In the first three frames, incorrect tiles can be seen on the border of the aggregate. These are inconsequential errors due to the equal on-rates of all tiles; they will fall off immediately and cause no permanent errors. However, in the last frame we see an incorrect tile which has been embedded within the aggregate; although it has a mismatch with its predecessors, successive tile additions have been correct *with respect to the error*, and now the erroneous tile has 3 matched edges. It has caused a permanent error, and the misinformation spreads to all downstream cells in the computation.

Array Size: In the last two frames, the size of the aggregate has exceeded the size of the array used in the simulation. Thus the Kinetic Assembly Model is not perfectly simulated; a maximal size of aggregate is imposed. In the simulations below, this does not affect the results in the region of interest, but it does explain the constant size (the maximum) found during fast, random aggregation.
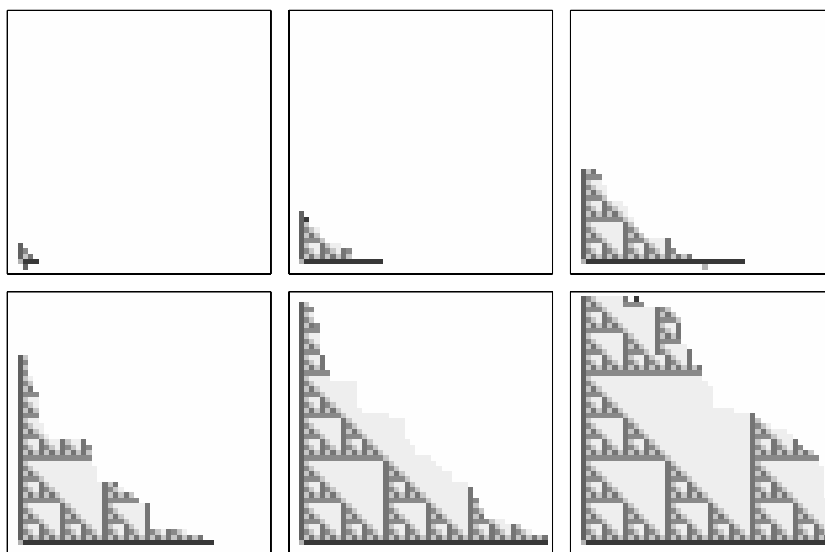
Figure 3.23: Growth of the Sierpinski Triangle. Greyscale indicates the tile type in the aggregate. The simulation uses parameters $G_{se} = 8$ and $\mathcal{T} = 1.95$, and the seed is a corner tile. These values correspond to monomer concentration of 3 μMand $r_f = 2$ /sec, with sticky ends of length 5, and $T = 45°$C; the frames show growth after 9, 18, 36, 63, 99, and 162 seconds.

To map out the parameter space of this model, simulations of the Sierpinski tiles were performed for all $1 \leq G_{mc}, G_{se} \leq 30$. Each simulation was run for $60/r_f$ simulated seconds, thus on average each unoccupied site could experience up to 60 on-events of each type; consequently, the distribution of aggregate sizes is comparable across different parameter values. Figure 3.24 shows the results for (a) aggregates seeded by the corner tile and (b) aggregates seeded by a rule tile, indicating both the resulting size of the aggregate and the number of errors[42] in the aggregate.

---

[42]What's actually calculated is the number of erroneous (mismatched) bonds, not the number of erroneous (incorrect with respect to their neighbors) tiles; a single misplaced rule tile could be responsible for 4 such mismatched bonds.

Figure 3.24: Phase diagrams for the Sierpinski tiles as computed by simulation: (a) aggregates seeded by the corner tile, and (b) aggregates seeded by a rule tile. Each disc represents the results of a single simulation on a 28×28 array; the size of the disc represents the final size of the aggregate,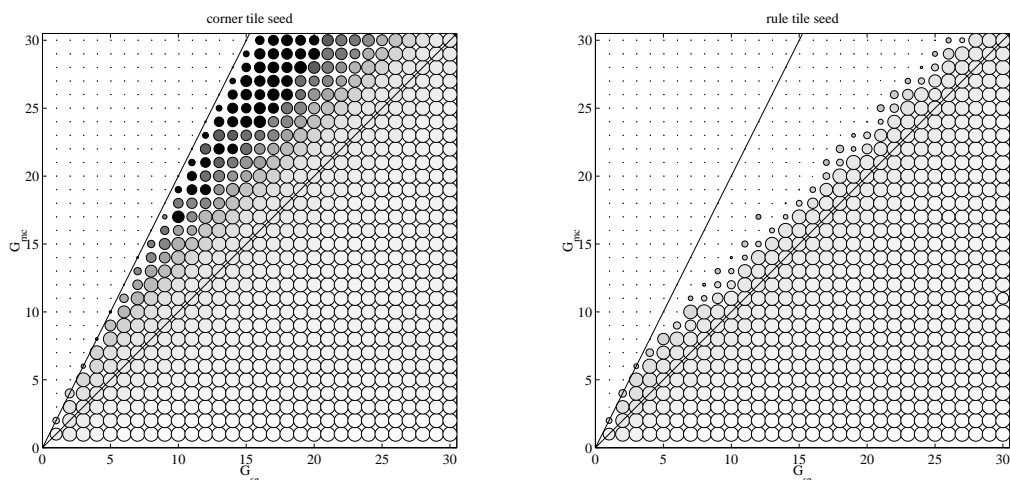 while the shading represents the number of errors as a fraction of total size. Each run was given the same "unitless" time; thus when G_{mc} is high (corresponding to low monomer concentration and thus slow assembly) more time is allowed so that error rates can be compared easily. Solid black indicates zero errors. We see three regimes: $\mathcal{T} > 2$ regime (no growth), $1 < \mathcal{T} < 2$ regime (includes error-free assembly near $\mathcal{T} = 2$), and $\mathcal{T} < 1$ regime (uncontrolled random growth to maximal size). Note that the $\mathcal{T} = 1$ transition is smooth, and hence is not a true phase boundary.

The lines show $\mathcal{T} = \frac{G_{mc}}{G_{se}} = 2$ and $\mathcal{T} = 1$, which we will respectively call the melting transition and the precipitation boundary. Above the melting transition, no aggregates grow from either seed. Below the precipitation boundary, monomers associate freely to produce random aggregates similar to those produces in the Tile Assembly Model at $\mathcal{T} = 1$. The rate of growth of random aggregates appears to fall off exponentially above the precipitation boundary; this is indicated by the decreasing size of aggregates seeded by a rule tile in (b) and by the decreasing error rate within aggregates seeded by the corner tile in (a). The result is that there is a large region of parameter space where simultaneously (1) growth does occur, (2) errors are rare, and (3) growth *not* initiated by the corner tile does *not* occur[43]. We call this *controlled growth*.

We are particularly interested in the behaviour of the Kinetic Assembly Model near the melting transition. Figure 3.25a shows the size and number of errors as a function of G_{se}, for $G_{mc} = 16$. Upon passing the melting transition ($G_{se} = 8$), the size of aggregates seeded by the corner tile grows dramatically, whereas aggregates seeded by the rule tile do not grow until $G_{se} \approx 12$, at which point all aggregates are overwhelmed with errors. There are a few isolated instances where aggregates seeded by the rule tile grow unusually large for G_{se} near 8; in these cases, the aggregate has incorporated a boundary or corner tile, which allows for further growth. Errors

However, at low error rates these two measures are equivalent. "100%" means 1 mismatched bond per tile; the error rate therefore could exceed 100% for optimally misplaced tiles, but it does not do so in these simulations.

[43]Starting with a boundary tile as a seed, growth would occur, but would soon incorporate a corner tile and produce a proper Sierpinksi triangle.
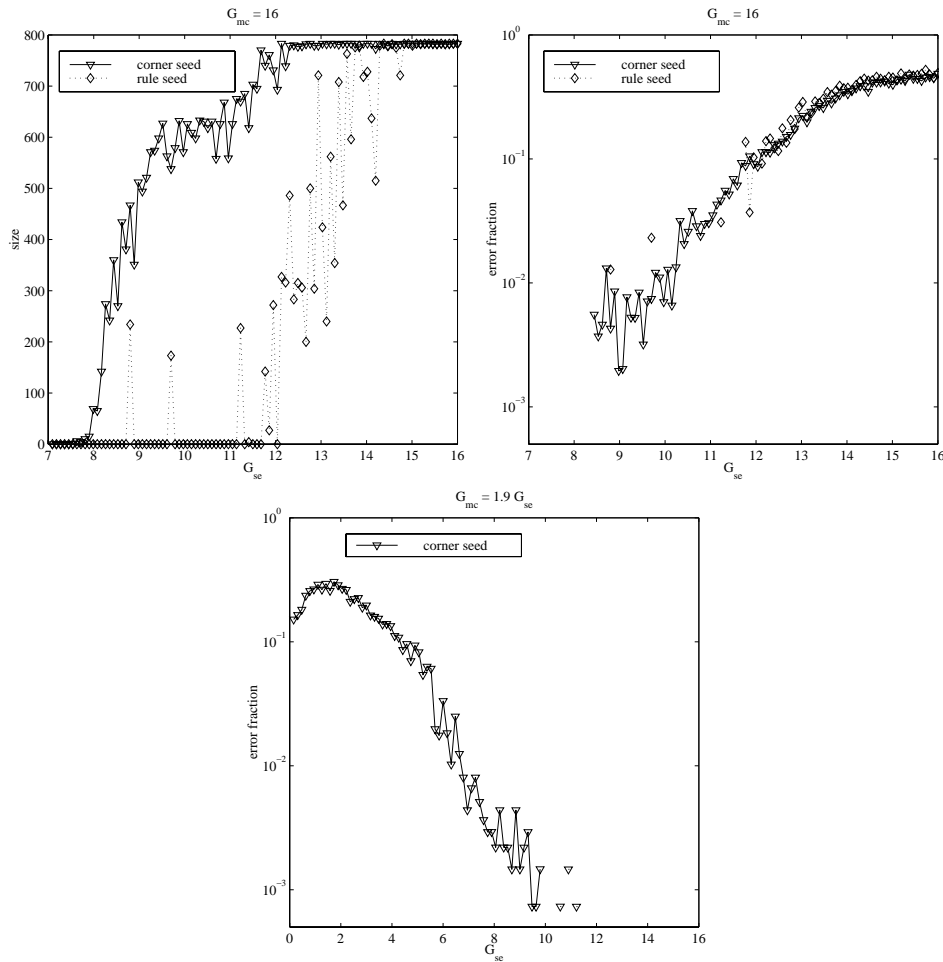
Figure 3.25: (a) Simulation results for $G_{mc} = 16$ for aggregates seeded with the corner tile and a rule tile. Note that for large G_{se}, where random aggregation is occurring, the aggregate grows to fill the entire 28×28 array. (b) Errors, as a fraction of aggregate size, along the line $G_{mc} = 16$. (c) Errors along the line $\mathcal{T} = 1.9$, using a 38×38 array. Because log axes are used, data points where the aggregate had zero errors are not shown.

appear to decrease exponentially as $G_{se} \to 8$ (Figure 3.25b). Figure 3.25c shows the behavior along $\mathcal{T} = 1.9$, where the system is sufficiently far below the melting transition to grow quickly, and yet sufficiently close to the melting transition to get low error rates; again, errors appear to fall exponentially with G_{se}.

In conclusion, it appears that with probability of error exponentially low in G_{se}, the kinetic model at $\mathcal{T} = 2 - \epsilon$ reproduces[44] the Tile Assembly Model at $\mathcal{T} = 2$.

---

[44]To account for the possibility that the Tile Assembly Model produces many distinct aggregates, we note that the probability that a size-n aggregate produced by the Kinetic Assembly Model is not also produced by the Tile Assembly Model is exponentially low.

3.3.5 Analysis

Equilibrium error rates. We would like to understand why the Kinetic Assembly Model produces these results. We begin by analyzing the equilibrium concentrations for the reaction equations. Consider an aggregate $A = T \cdot A'$ where the tile T has b bonds with A'. At equilibrium, the principle of detailed balance tells us that[45]

$$\frac{[A]}{[A'][T]} = \frac{k_f}{k_r} \quad \text{thus} \quad \frac{[A]}{[A']} = \frac{k_f[T]}{k_r} = \frac{r_f}{r_{r,b}} = e^{-(G_{mc} - bG_{se})}.$$
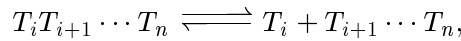
Calculating equilibrium concentrations from any order of tile addition steps yields the same result, so we can calculate the concentration of $A = T_1 T_2 \cdots T_n$ from any sequence of additions for producing A. Let b_i be the number of bonds for the addition

$$T_i T_{i+1} \cdots T_n \; \xrightleftharpoons{\hspace{1cm}} \; T_i + T_{i+1} \cdots T_n,$$

and let $b_A = \sum_{i=1}^{n-1} b_i$ be the total strength of all matching edges in the aggregate. Then,

$$
\begin{aligned}
\frac{[A]}{[T]} &= \frac{[T_1 \cdots T_n]}{[T_n]} = \frac{[T_1 \cdots T_n]}{[T_2 \cdots T_n]} \frac{[T_2 \cdots T_n]}{[T_3 \cdots T_n]} \cdots \frac{[T_{n-1} \cdots T_n]}{[T_n]} \\
&= e^{-(G_{mc} - b_1 G_{se})} e^{-(G_{mc} - b_2 G_{se})} \cdots e^{-(G_{mc} - b_{n-1} G_{se})} \\
&= e^{-((n-1)G_{mc} - b_A G_{se})} = e^{-((n-1)\mathcal{T} - b_A)G_{se}}.
\end{aligned}
$$

So we see that the concentrations of aggregates with $\frac{b_A}{n-1} > \mathcal{T}$ will grow with n, while the concentrations of other aggregates will shrink[46]. We would like to make a prediction for error rates based on the equilibrium assumption. To do this, we ignore the total concentration, and just ask, "Of all material containing size n aggregates, what fraction is without errors?"

To compute this, we must know the value of b_A for aggregates of interest. Note that for the Sierpinski Tiles, any aggregate A_0 produced by the Tile Assembly Model at $\mathcal{T} = 2$ (i.e., an aggregate with 0 errors) has exactly $b_{A_0} = 2(n-1)$ because every tile addition step contributes exactly 2 bonds. Furthermore, all other aggregates must have $m \doteq 2(n-1) - b_A > 0$, a measure of their suboptimality[47]. Aggregates with small m look like perfect Sierpinski aggregates, but with a few internal errors. For size n aggregates, one perfect and one suboptimal by m,

$$\frac{[A_m]}{[A_0]} = \frac{e^{-((n-1)\mathcal{T} - b_{A_m})G_{se}}}{e^{-((n-1)\mathcal{T} - b_{A_0})G_{se}}} = e^{-mG_{se}}.$$

This at least partly explains the absence of aggregates seeded by rule tiles: any aggregate consisting entirely of rule tiles must have $m \geq 2\sqrt{n} - 2$, and thus their equilibrium concentrations are

---

[45]Note that r_f is constant because all monomer concentrations are equal and held constant, while $r_{r,b}$ depends on b for the particular reaction.

[46]Recall that we are assuming equilibrium has been reached; taken literally, this is patently absurd when at equilibrium the concentrations of aggregates grows exponentially with their size. The implication is that in order to hold the monomer concentrations constant, we must continually be providing new material into the system; this new material flows through the system to create larger and larger aggregates.

[47]This can be seen by noting that $b_A \leq 2\#(rule\ tiles + corner\ tiles) + 2.5\#(boundary\ tiles)$ where the deficit is due to internal mismatches and to the "surface energy" of unmatched edges on the perimeter. An aggregate consisting exclusively of n rule tiles will have perimeter at least $4\sqrt{n}$, and thus $b_A \leq 2n - 2\sqrt{n}$ and $m \geq 2\sqrt{n} - 2$. An aggregate with $g + 1$ boundary and corner tiles will have 2 mismatched or unmatched edges terminating the boundary line and on the perimeter at least g umatched edges; thus $m \geq 0$.

exceedingly low[48].

To compute the fraction of all size-n material which is errorless, we must know *how many* aggregates of each kind there are. Let n_m be the number of distinct size n aggregates of sub-optimality m. Then a size n aggregate chosen from the equilibrium distribution is errorless with probability

$$\Pr_{eq}(errorless\ aggregate|n) = \frac{n_0[A_0]}{\sum_{m=0}^{\infty} n_m[A_m]} = \frac{1}{\sum_{m=0}^{\infty} \frac{n_m}{n_0} e^{-mG_{se}}}.$$

For small m, we can estimate $\frac{n_m}{n_0}$ by noting that for each perfect aggregate of size n, we can make $\approx \binom{2n}{m}$ suboptimal aggregates by inducing errors at m internal edges, and completing the rest of the pattern properly. Thus,

$$\Pr_{eq}(errorless\ aggregate|n) \approx \frac{1}{\sum_{m=0}^{\infty} \binom{2n}{m} e^{-mG_{se}}} = \frac{1}{(1 + e^{-G_{se}})^{2n}} \approx 1 - 2ne^{-G_{se}}.$$

We can see that $\Pr_{eq}(errorless\ aggregate) \approx \frac{1}{e}$ at $n = \frac{1}{2}e^{G_{se}}$. Since the G_{se} is determined by the length of sticky ends, we see that by increasing sticky end length, we can exponentially increase the size over which errorless computation can be expected to occur.

We could have arrived at the same conclusion more simply, but less rigorously, by assuming that all tile additions occur in slots and the tile is chosen independently from the local equilibrium distribution. (A site is in *local equilibrium* when the tiles (or their absence) at neighboring positions do not change, and all tile addition and tile dissociation reactions involving the site are in equilibrium.) Then,

$$\Pr_{eq}(errorless\ aggregate|n) \approx \Pr_{eq}(errorless\ step)^n \approx \left(\frac{1}{1 + 2e^{-G_{se}}}\right)^n \approx 1 - 2ne^{-G_{se}}.$$

Note that this analysis, based on assumptions of equilibrium, predicts that error rates are unaffected by G_{mc}. This was not the result of our simulation: error rates increase dramatically as G_{mc} drops below the melting transition (i.e., as monomer concentration increases). Consequently, we conclude that equilibrium is *not* achieved in these cases.

The kinetic trap. What prevents the system from achieving equilibrium? The intuition is that if the growth of the crystal is faster than the time required to locally establish equilibrium at the growth sites, tiles will become embedded and "frozen" in the interior of the aggregate with an out-of-equilibrium distribution.

How long does it take for a growth site to reach local equilibrium? Consider a growth site that has just formed, and assume that the local context (neighboring tiles) does not change. Monomer tiles of all kinds will sit down at the site, stay a while, and then leave, each according to its own off-rate. If we look immediately after the growth site appears, the probability that the site is empty is near 100%; however, if we wait a very long time before looking, we will find each tile, or an empty site, with their equilibrium probabilities. If the local context *does* change by addition of tiles surrounding the growth site, then the tile currently in place can be "frozen" there effectively permanently; even if it has one mismatched edge, three matches on its other edges can make its off-rate very low. Although this is a very cartoonish picture, it is the basis for our analysis, since the full system is too complex to treat rigorously.

---

[48]The concentration of a rule tile aggregate A_r is bounded by $[A_r]/[T] \leq e^{(\mathcal{T}+\epsilon n - 2\sqrt{n})G_{se}}$, which has a minimum of $[A_r]/[T] \leq e^{(\mathcal{T}-1/\epsilon)G_{se}}$ at $n_{critical} = \epsilon^{-2}$. (Recall that $\mathcal{T} = 2 - \epsilon$.) The concentration at the critical size, which
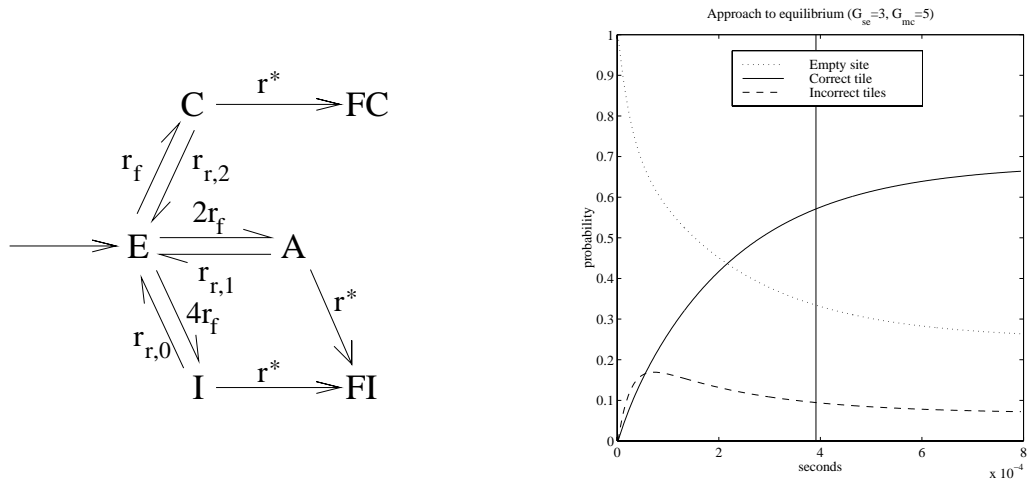
Figure 3.26: Model for kinetic trapping at a single growth site. (a) Simplified model for the filling of a new site. In state E the site is empty; in state C a correct tile is present; in state A an almost correct tile (with one mismatch) is present; and in state I a tile with several mismatches is present. The sinks FC and FI represent frozen correct tiles and frozen incorrect tiles, respectively. (b) The approach to equilibrium distribution at the site, assuming the site has not yet been frozen. The vertical bar marks the expected time at which the site will be frozen.

Let's look at the probability of a particular tile being present in the site as a function of time, prior to the site being frozen. For the Sierpinski Tiles, four cases must be distinguished: (E) The site is empty. The "off-rate" of emptiness is $7r_f = 7k_f e^{-G_{mc}}$, since there are 7 tiles. (C) The correct tile is in place. It's off-rate is $r_{r,2} = k_f e^{-2G_{se}}$. (A) One of two tiles with just one match, and off-rate $r_{r,1} = k_f e^{-G_{se}}$. (I) One of 4 tiles with no matches, and off-rate $r_{r,0} = k_f$. Let $p_i(t)$ be the probability that (i) is the case t seconds after the growth site has appeared, assuming the site has not yet been frozen. The rate equations for the model in Figure 3.26a, excluding the sinks FC and FI, can be written as

$$\dot{\mathbf{p}}(t) = \begin{bmatrix} -7r_f & r_{r,2} & r_{r,1} & r_{r,0} \\ r_f & -r_{r,2} & 0 & 0 \\ 2r_f & 0 & -r_{r,1} & 0 \\ 4r_f & 0 & 0 & -r_{r,0} \end{bmatrix} \begin{bmatrix} p_E(t) \\ p_C(t) \\ p_A(t) \\ p_I(t) \end{bmatrix} \doteq M\mathbf{p}(t)$$

and thus, $\mathbf{p}(t) = e^{Mt}\mathbf{p}(0)$ where $\mathbf{p}(0) = [1\ 0\ 0\ 0]^T$.

The behavior of $\mathbf{p}(t)$ is shown in Figure 3.26b. We want to know the probability that the correct tile is in place when the site is frozen. During controlled growth, the rate of growth is approximately $r^* = r_f - r_{r,2}$; thus a given site will be frozen in mean time approximately $t^* = 1/(r_f - r_{r,2})$. With a decrease in G_{mc} (increased monomer concentration), r_f increases, and t^* becomes earlier, leading to a more out-of-equilibrium frozen distribution.

By including sink states FC and FI into the model of Figure 3.26a, we can solve exactly for

---

becomes a kinetic barrier to the formation of larger aggregates (Erickson 1980), approaches zero as $\mathcal{T} \to 2$.

the frozen distribution. In this case the equations are

$$
\dot{\mathbf{p}}(t) =
\begin{bmatrix}
-7r_f & r_{r,2} & r_{r,1} & r_{r,0} & 0 & 0 \\
r_f & -r_{r,2}-r^* & 0 & 0 & 0 & 0 \\
2r_f & 0 & -r_{r,1}-r^* & 0 & 0 & 0 \\
4r_f & 0 & 0 & -r_{r,0}-r^* & 0 & 0 \\
0 & r^* & 0 & 0 & 0 & 0 \\
0 & 0 & r^* & r^* & 0 & 0
\end{bmatrix}
\begin{bmatrix}
p_E(t) \\
p_C(t) \\
p_A(t) \\
p_I(t) \\
p_{FC}(t) \\
p_{FI}(t)
\end{bmatrix}
\doteq M\mathbf{p}(t)
$$

where $\mathbf{p}(0) = [1\ 0\ 0\ 0\ 0\ 0]^T$. The probability of the site being frozen with the correct tile, $p_{FC}(\infty)$, can be easily computed from the steady-state of the related flow problem, where a unit amount of material is pumped into state E and accumulates differentially in FC and FI:

$$
\dot{\mathbf{p}}(\infty) = [1\ 0\ 0\ 0\ p_{FC}(\infty)\ p_{FI}(\infty)]^T = M\mathbf{p}(\infty).
$$

A little algebra gives the probability of an errorless step in terms of G_{se} and G_{mc}:

$$
\mathrm{Pr}_{kin}(errorless\ step) \doteq \mathbf{p}_{FC}(\infty) = \frac{\frac{1}{r^*+r_{r,2}}}{\frac{1}{r^*+r_{r,2}} + \frac{2}{r^*+r_{r,1}} + \frac{4}{r^*+r_{r,0}}}.
$$

In this equation for errors due to kinetic trapping, in contrast to the equilibrium prediction, the error rate depends upon *both* G_{se} and G_{mc}. The equation predicts error rates that are in qualitative agreement with the simulations, as shown in Figure 3.27. In this analysis, it becomes clear that in the limit as $\mathcal{T} \to 2$ and thus $r^* \to 0$,

$$
\mathrm{Pr}_{kin}(errorless\ step) \to \mathrm{Pr}_{eq}(errorless\ step) = \frac{1/r_{r,2}}{1/r_{r,2} + 2/r_{r,1} + 4/r_{r,0}} \approx \frac{1}{1+2e^{-G_{se}}}.
$$

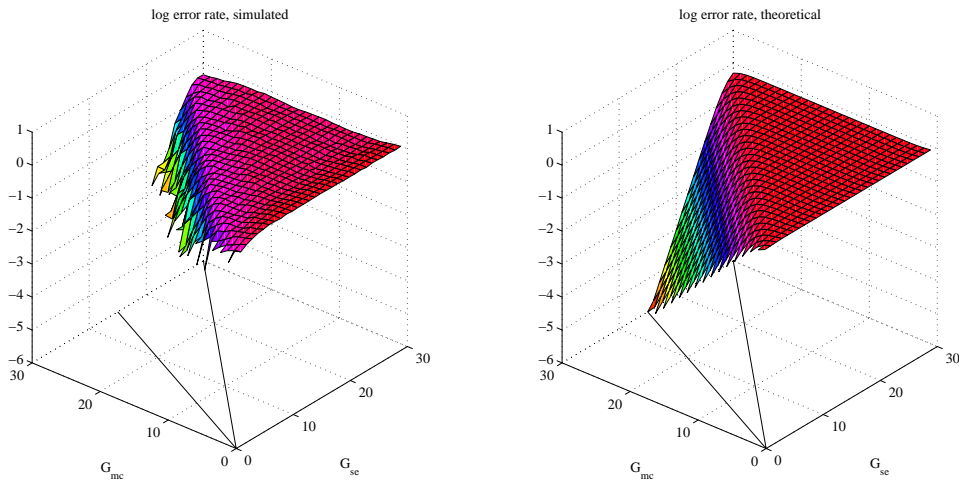Thus equilibrium error rates are achieved near $\mathcal{T} = 2$.

Figure 3.27: (a) Log$_{10}$ per-step error rates, estimated from simulations. (b) Log$_{10}$ per-step error rates, according to the kinetic trap theory.

Speed of assembly. We have already observed that the forward rate $r_f = \hat{k}_f e^{-G_{mc}}$ depends upon monomer concentration, and consequently, as our error rates improve with increased G_{mc}, simultaneously the speed of computation drops dramatically. Now that we have an analytical expression for $\mathrm{Pr}_{kin}(errorless\ step)$, based upon our simplified kinetic trap model, we can determine the conditions which achieve a given target error rate ε with the fastest rate of assembly $r^* = r_f - r_{r,2}$:

$$1 - \varepsilon = \mathrm{Pr}_{kin}(errorless\ step) \approx \frac{1}{1 + 2\frac{r^* + r_{r,2}}{r^* + r_{r,1}}}$$

and thus for small ε and $2G_{se} > G_{mc} > G_{se}$,

$$\varepsilon \approx 2\frac{r^* + r_{r,2}}{r^* + r_{r,1}} \approx 2e^{-(G_{mc} - G_{se})} \doteq 2e^{-\Delta G}.$$

To achieve error rate ε, the system can be run anywhere along a line parallel to $\mathcal{T} = 1$ but displaced by $\Delta G = \ln 2/\varepsilon$ above it. Where along this line does self-assembly proceed most rapidly? We find the maximum of

$$r^* = \hat{k}_f(e^{-G_{mc}} - e^{-2G_{se}}) = \hat{k}_f(e^{-\Delta G - G_{se}} - e^{-2G_{se}})$$

by differentiation with respect to dG_{se}; optimal growth for constant ΔG occurs at

$$G_{se} = \Delta G + \ln 2 = \ln\frac{4}{\varepsilon} \quad \text{and} \quad G_{mc} = 2\Delta G + \ln 2 = \ln\frac{8}{\varepsilon^2}.$$

The optimal growth rate $r^* = \frac{\hat{k}_f}{16}\varepsilon^2 \approx 0.75 \times 10^6\ \varepsilon^2\ /sec$ occurs on the line $G_{mc} = 2G_{se} - \ln 2$.

Thus it appears that we have a hard physical limit on what error rates can be achieved by DNA self-assembly within reasonable time limits. If we wish to have a net forward rate of 1 tile added per second, then the best we can achieve is an error rate of 1/1000; while if we were willing to wait half an hour for each addition, we could get an error rate of 3×10^{-5}, and we could grow some perfect 200×200 aggregates over the course of a week.

3.3.6 Discussion

The above simulations and theoretical arguments both confirm that in the Kinetic Assembly Model, aggregates can grow with finite speed and arbitrarily low per-site error rates for large G_{se} and $\mathcal{T} = 2 - \epsilon$. We should be careful that the analysis does not depend upon the particularities of the Sierpinski Tiles. It can easily be verified that if the rule tiles use k labels (instead of the 2 labels used in the Sierpinski Tiles) and there are a total of N tiles (instead of the 7 Sierpinski Tiles) then the analysis is unchanged except that

$$\mathrm{Pr}_{eq}(errorless\ aggregate|n) \approx 1 - 2(k - 1)ne^{-G_{se}}$$

and

$$\mathrm{Pr}_{kin}(errorless\ step) = \frac{\frac{1}{r^* + r_{r,2}}}{\frac{1}{r^* + r_{r,2}} + \frac{2(k-1)}{r^* + r_{r,1}} + \frac{N - 1 - 2(k-1)}{r^* + r_{r,0}}} \approx 1 - 2(k - 1)e^{-(G_{mc} - G_{se})}$$

and the optimal growth rate now occurs displaced $\Delta G = \frac{\ln 2(k-1)}{\varepsilon}$ above $\mathcal{T} = 1$. We now loosely discuss other aspects of the model.
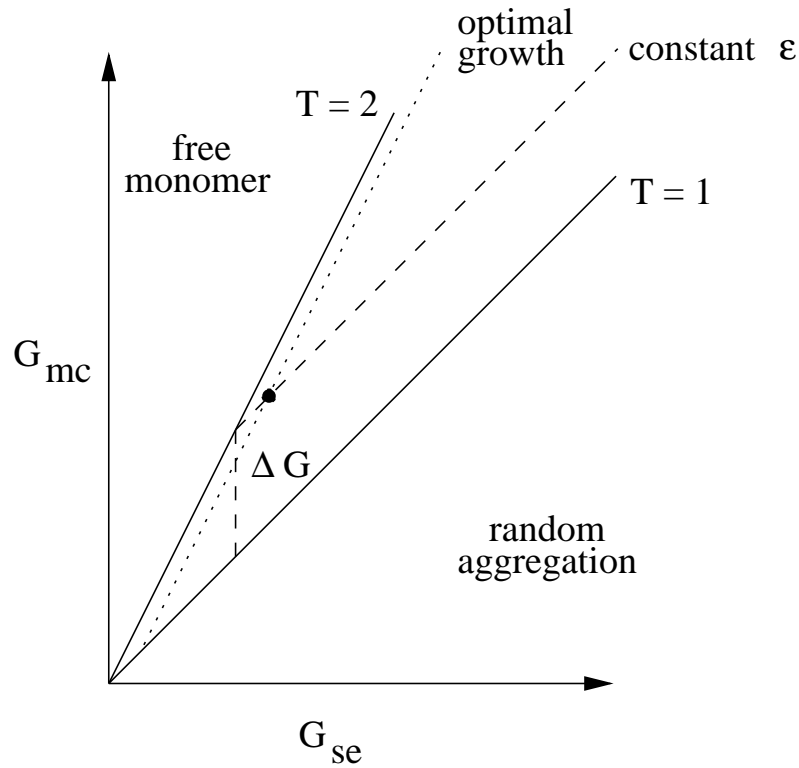
Figure 3.28: Analysis of the phase diagram for 2D self-assembly. Lines mark the melting transition $\mathcal{T} = 2$, the precipitation boundary $\mathcal{T} = 1$, a line of constant error rate $G_{mc} = G_{se} + \Delta G$, and the line on which optimal growth rates occur $G_{mc} = 2G_{se} - \ln 2$.

Energy use. Reversible computers have the potential to compute using arbitrarily little energy per step, because no information is erased during the computation itself (Landauer 1961; Bennett 1973). The system described here uses only fully physically reversible reactions, and thus is a candidate for low-energy computation; although non-reversible 1D cellular automata may be simulated, the 2D pattern records a history of the entire computation, and thus no information is lost at any step. During controlled growth at $\mathcal{T} = 2 - \epsilon$, the amount of energy used by the system equals the free energy lost as heat on each step:

$$-\Delta G^{\circ} = -(G_{mc} - 2G_{se})RT = \epsilon G_{se}RT.$$

For any fixed G_{se}, error rates and energy use are simultaneously minimized as the melting transition is approached.

An entropic ratchet. What happens at $\mathcal{T} = 2$ exactly? We already know that at $\mathcal{T} = 2$, optimal equilibrium error rates are achieved and no energy is used to power each step; the probability of going backwards is identical to the probability of going forwards. In a 1D reversible computation, like that imagined by Bennett (1973), the random walk would lead to no net computation performed. However, in our 2D system, the number of possible errorless size n aggregates grows with n. Thus, as the state-space is explored at equilibrium, it will be entropically driven to perform computation! This oddity deserves further attention to see whether it would still be present in a

more realistic model.

Experimentally accessibly regimes. We have already developed the relation between our abstract parameters G_{mc} and G_{se} and relevant parameters of a real system, such as monomer concentration and free-energies of hybridization; Figure 3.23 showed that low error rates can be achieved for realistic parameters[49], given our assumptions. We can make our arguments more realistic by considering what happens as a solution of monomers is slowly annealed from a high temperature to a lower temperature. At any moment in time, we plot the current reaction conditions as a point on Figure 3.24 to determine the rate of growth and per-step error rate. Suppose initially $G_{mc} = 12$ and $G_{se} = 5$; here, above the melting transition, the monomers are all free in solution. As the temperature decreases, G_{se} will increase, and our point follows a horizontal trajectory straight toward $\mathcal{T} = 2$. Just below the melting transition, the aggregate will grow (with optimal error rates for the current G_{se}). Consequently, the monomer concentration will drop, and G_{mc} will increase, bringing the system back toward $\mathcal{T} = 2$. So long as the temperature drops slowly enough, the system will stay just below the melting transition, and our point will follow a trajectory parallel to $\mathcal{T} = 2$. Thus, by annealling, the self-assembly process will automatically maintain itself in the regime where errors are most infrequent. Optimal annealing schedules are an issue for future investigation, and to be of practical use they will have to take account of the non-idealities of the system.

Imperfections of a real system. The careful reader will immediately observe that the concentrations of different tiles will be depleted at different rates, thus breaking our original assumption that all tiles are present at equal concentrations. This will introduce additional factors into the error analysis. There are many other ways in which real systems will deviate from the Kinetic Assembly Model. Free energies of hybridization for different sticky-end sequences cannot be perfectly matched, so the melting transitions for different tiles will differ slightly. Worse yet, imperfectly or partially matched sticky-ends may contribute to the free energy of interaction between tiles with mismatched edges, in violation of the model's assumption that only correctly edges contribute to $\Delta G^{\circ}_{b.s}$. It remains to be determined how important these factors are.

Cooperativity of binding. The Kinetic Assembly Model makes a strong assumption that two binding sites on the same tile will act cooperatively when binding to an aggregate. Specifically, it is claimed that $\Delta G^{\circ}_{2\ bonds} = 2\Delta G^{\circ}_{1\ bond}$. There are three points to make. First, the rigidity of double crossover molecules, as demonstrated by Li et al. (1996), suggests that the binding events should act together – in particular, the slot-filling event during proper growth should be cooperative. This intuition can be bolstered by estimating the "effective" local concentration of the remaining sticky end after one end has bound – giving an estimate for the additional "loop entropy"(Cantor and Schimmel 1980, p. 1205) required to close the second end in the slot. Since double-stranded DNA has a persistence length of approximately 130 nt (Cantor and Schimmel 1980, p. 1033) and DX molecules span roughly 40 nt from end to end, the physical distance between the sticky ends may fluctuate from 12 to 14 nm, thus exploring a volume of ≈ 4000 nm^3, with the free sticky end assuming perhaps a range of $30° \times 30°$ orientations at each position. This corresponds to an effective concentration of

$$C_{eff} = \frac{1\ \text{sticky end}}{4000\,\text{nm}^3 \times 900\ deg^2}\frac{(10^{24}\ \text{nm}^3 \times 360^2\ deg^2)\,/\text{liter}}{6 \times 10^{23}\ \text{sticky ends/mol}} = 60\,\text{mM}$$

and thus a loop entropy $\Delta S_{loop} = R \ln C_{eff} = -5.6$ cal/mol/K. This value is comparable with the initiation entropy of $\Delta S_{init} = -6$ cal/mol/K. At $27°$C it increases the free energy of interaction by

---

[49]$G_{mc} = 30$ is an example of an unrealistic parameter: at 2 pM, $r_f = 2 \times 10^{-5}$ /sec and monomer addition will occur only twice per day.

1.68 kcal/mol, which roughly offsets the contribution of a single base-pair bond (-1.4 kcal/mol). The deviation from perfect cooperativity should be negligible, according to this estimation. Experimental studies should be able to measure the extent of cooperativity; the preliminary experiment reported in Winfree et al. (in press) argues qualitatively for cooperativity in an analogous DNA system.

Second, it is possible that in addition to free energy due to sticky end hybridization and due to loop entropy, there could be enthalpic contributions to loop closure, for example, if the double helix must be twisted, stretched, or otherwise deformed in order to fit into the slot. Double crossover molecule tiles can be designed with the intention of minimizing these anticooperative effects, but it remains to be seen how well that works. It may also be possible to *exploit* anticooperative effects to enforce *negative interactions* for mismatching edge labels. This would require using differently sized double crossover molecules, for example by changing the lengths of the four arms, so that geometric mismatches are present in addition to sticky-end sequence mismatches. It may be possible this way to implement a tile assembly model with negative weights.

Third, just as the initiation entropy was folded into the abstract G_{mc} and G_{se} parameters, a loop entropy or mild anticooperative adjustment could be taken up by adjusting G_{mc} and G_{se} to reproduce the on-rates and off-rates for the most important double-match and single-match cases. The simple model would be inaccurate for the off-rates of tiles with more than 2 bonds, but as these tiles seldom dissociate for parameters of interest, this inaccuracy is irrelevant.

Alternative reaction mechanisms. The Kinetic Assembly Model assumes that the growth of aggregates occurs by addition of single monomers only, and thus that there are no interactions between aggregates. Reaction mechanisms would not affect the equilibrium error rate predictions, but Rothemund (personal communication) has emphasized that dimer-dimer pathways, or other interactions between aggregates, could be very important for the kinetics of self-assembly, and thus their inclusion could affect kinetic trapping in theory and in practice. Indeed, Malkin et al. (1995) have directly observed, by AFM, crystal growth by sedimentation of small three-dimensional nuclei.

It is also possible – perhaps I should say probably – that alternative reaction mechanisms are present for creating non-planar structures, such as tubes or random three dimensional networks. Indeed, experimental studies attempting to create 2D lattices of DX molecules (Winfree et al. 1998) found, for example, occasional unexpected rod-like structures in addition to the expected planar 2D crystals.

3.3.7   Conclusions

We have used a pair of simple kinetic models to understand error rates in the self-assembly process for algorithmically-defined 2D polymerization. Our results lend credence, in lieu of a full experimental demonstration, to proposals (Winfree 1996b; Winfree et al. in press) for computation by self-assembly of DNA: we have found that 2D self-assembly can theoretically support computation with arbitrarily low error rates. This answers a question raised by Reif (in press), who was concerned that, as in the $\mathcal{T} = 1$ example of Figure 3.19, an unfortunate sequence of tile additions could lead to *blockages* where no tile can fit into an empty site without a mismatch. We find that blockages are not a problem in our model, but the thermodynamics of DNA hybridization give rise to an intrinsic per-step error rate. Large computations require low concentrations and hence very slow growth rates. This is the algorithmic equivalent of the fact, in conventional crystallization, that large perfect crystals form under conditions of slow growth near the solubility line (Kam et al. 1980).

A few worked-out examples for the case of the Sierpinski Tiles are illustrative. From our investigations of kinetic trapping, we found that there is an optimal growth rate $\hat{r}^*$ for every target error rate ε. At this growth rate $\varepsilon = 4e^{-G_{se}}$, $[\text{DX}] = 2.5\varepsilon^2$ M, and $r^* = 0.75 \times 10^6 \varepsilon^2$ /sec, where $0 < G_{se} = (\frac{4000\,\text{K}}{T} - 11)s \approx -\Delta G^\circ / RT$ for the hybridization of a single sticky end of length s. Assemblies of $n_{max} \doteq 1/\varepsilon$ tiles would be expected to contain one error on average; there is an inverse relationship between the rate of assembly and the expected size of error-free aggregates. For example, sticky ends of length 5 at room temperature give $G_{se} = 12$ and $n_{max} = 40000$, but requires a concentration of $[\text{DX}] = 1.5\,\text{nM}$ and thus a rate $r^* = 1.6$ /hour. The same system could be run at 17°C, where $G_{se} = 14$, $[\text{DX}] = 30$ pM, $n_{max} = 300000$, and $r^* = 0.71$/day; or at 45°C, where $G_{se} = 8$, $[\text{DX}] = 4.5\,\mu\text{M}$, $n_{max} = 750$, and $r^* = 1.35$ /sec. Under the latter conditions, a non-deterministic set of DNA tiles in a reasonable volume (1 ml) could give rise to 10^{13} distinct 300-tile aggregates in under a minute, that is, 10^{14} operations per second. This would be sufficient for solving a simple 40-variable SAT problem by subsequent ligation and PCR to find the answer-containing strand in the "good" aggregate. However, for this application an additional source of errors would be false-positives due to non-answer aggregates which, because of an error during assembly or during PCR, appear to be "good;" an additional error analysis is required in this case.

What are we to do if we want faster and less error-prone computation? Reif (in press) suggests using a combination of autonomous self-assembly and step-wise processing; his ingenious constructions perform a computation in a series of self-assembly steps each of which only requires the formation of small aggregates. Because the number of steps is kept low (for example, computing a circuit of size s requires $O(\log s)$ self-assembly steps), there is promise for asymptotically better error rates; however, a detailed analysis remains to be done, and may be difficult due to the lack of experimental evidence for the complex DNA structures and self-assembly reactions he proposes.

Is it possible to get faster and less error-prone computation in an autonomous self-assembly system? Biology makes use of an energy source to improve error rates by "proofreading" mechanisms (Kornberg and Baker 1991). Kinetic proofreading mechanisms can be fairly simple (Hopfield 1974); it would be interesting if such a mechanism could be devised to mediate the self-assembly of double-crossover molecules. Alternatively, one can accept the intrinsic error rate and try to devise error-correcting *algorithms* which could improve the overall error rate exponentially with a slowdown only linear in the number of extra tile types.

Chapter 4 Experiments with DNA Self-Assembly

4.1 A Competition Experiment: Slot-Filling

Abstract[1] In this section we examine the question of whether the two binding domains in the input region of a rule molecule act cooperatively during a slot-filling reaction. A well-defined DNA system has been designed to model a single slot analogous to the one in a growing lattice. The system consists of three logical pieces: ABC, D, and D'. D and D' model rule molecules which match either both or just one of the slot's two sticky ends. Thus, by competing D again D', we can determine the extent of the preference for D over D' for filling the slot. By analyzing ligation products, wherein a closed circular species indicates the correct insertion of D in the slot, we observed that D was preferred even in the presence of a 64-fold excess of D'. This is strong suggestive evidence that the slot-filling reaction in lattice formation is also cooperative, as is required for our model of computation by self-assembly.

The theoretical studies of previous chapters only point to the possibility of algorithmically-patterned lattices of DNA. There are two major issues to be investigated experimentally. The first is whether homogeneous lattices will form; i.e., whether the geometric structure itself will self-assemble. The second issue is whether, in the presence of multiple units in solution, the logically correct unit will hybridize in each slot. This competitive process for filling each slot is essential for computation, as a single error can propagate throughout the entire computation. Ultimately, error rates will determine the size of lattice in which reliable computation may be performed.

We have begun an investigation of the second issue. We first build a model molecular complex, called ABC, which contains a single slot and no other sticky ends. ABC is composed of two double crossover molecules, A and C, and a duplex linker B. ABC is created by ligating eight oligonucleotides; the final structure contains four hybridized strands. Rather than test the assembly of a double crossover unit into ABC's slot, we model the unit by a linear duplex "linker", called D. When ABC is properly hybridized to D, we call the complex ABCD. Completely ligated, ABCD is a complex catenane with four interlocked circles. To test the specificity of the hybridization, we also have a mismatched linker D', which is perfectly complementary to only one of the sticky ends in the slot. We expect that ABCD' cannot be completely ligated, due to the mismatch, and hence ABCD' does not form a catenane. These molecular complexes are diagramed in Figure 4.1.

Experimentally, we must establish that the double crossover molecules A and C form properly upon annealing their component strands. As developed in Fu and Seeman (1993), where the details of hybridization were probed by more extensive structural characterization, a good indication of proper association is a single band of mobility in a non-denaturing gel appropriate for the topology and molecular weight. The ligation products ABC, ABCD, and ABCD' (by which we mean whatever

```
A1  (88 nt) = TAATGTGCCTGACCGCCTTACTTTTGTAAGGCGGTCACCGAATTCCGACTTTTGTCGGAATTCGGACGCCTAACGTGGACACCGCGAC
A2  (52 nt) = GTAAGCTTCCTGTCCACGTTAGGCGTGGCACATTAGTCGCGGACTTAGACAA
A3  (32 nt) = GGATGGTTGTCTAAGTGGAAGCTTACCGCATC
B1  (64 nt) = CCATCCAGCGTTGACACGTCTGCGTAACTCACGGTAGTGTAAACCTTGATTGAATCGGCAGTAC
B2  (52 nt) = CCGATTCAATCAAGGTTTACACTACCGTGAGTTACGCAGACGTGTCAACGCT
C1  (80 nt) = TGCATCTGGACCTCGAGACTTTTGTCTCGAGGTGGTTCGCCGCTTTTGCGGCGAACCTGAACCACGAACGTGGTGGATCAC
C2  (52 nt) = GAGTCGACGGACCACGTTCGTGTTCACCAGATGCAGTGATCCTGCATATGAC
C3  (32 nt) = CTGAGCGTCATATGCACCGTCGACTCGTACTG
D1  (64 nt) = GCTCAGCCGTGCTAATCCAACTCGGTACCTACAGATACGATGGACTGGTTAGATAGGTGATGCG
D2  (52 nt) = ACCTATCTAACCAGTCCATCGTATCGTAGGTACCGAGTTGGATTAGCACGG
D1' (64 nt) = GCTCAGCCGTGCTAATCCAACTCCTGCAGTACAGATACGATGGACTGGTTAGATAGGTCAACAG
D2' (52 nt) = ACCTATCTAACCAGTCCATCGTATCTGTACTGCAGGAGTTGGATTAGCACGG
```
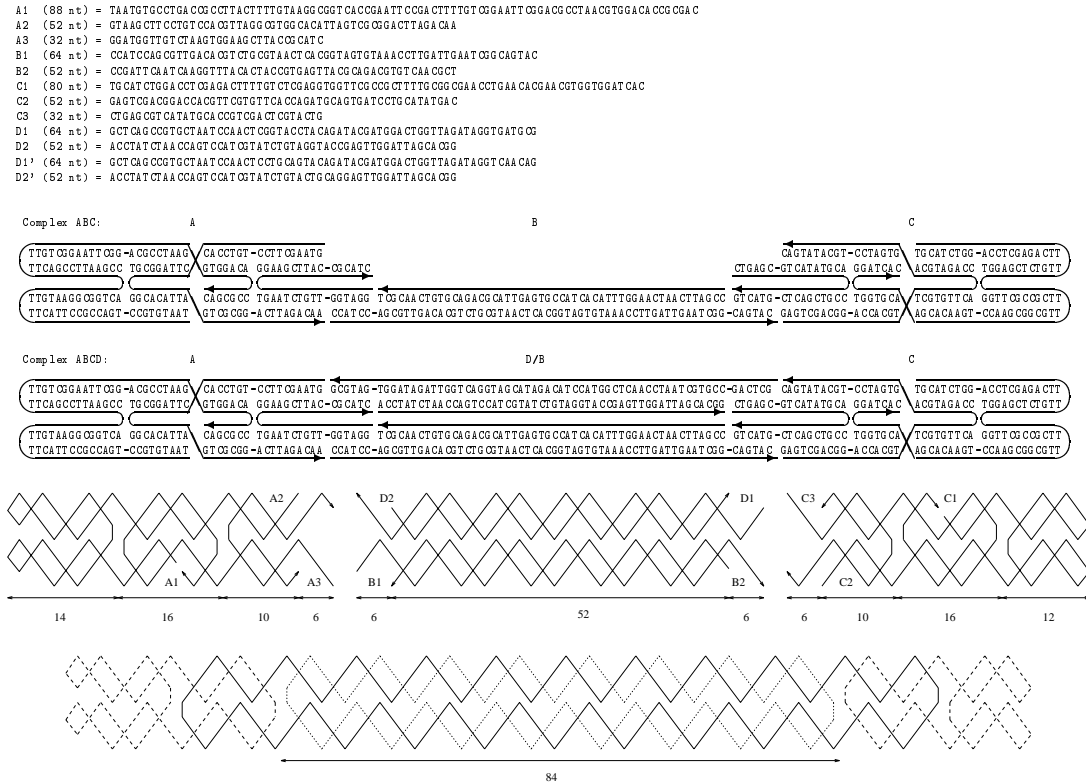

Figure 4.1: Sequences and Structures for ABCD. Sequences are written 5' to 3'. 3' ends are denoted by arrows in the diagrams, and strand labels are near 5' ends. Lengths are measured in nucleotides. Above, sequence details are given along with schematic representations of ABC and ABCD (not ligated). Below, more geometric detail is sketched for A, B, C, D, and ABCD (ligated). Diagrams illustrate intended structures only.

it is we get when we intend to make structures ABC, ABCD, and ABCD' respectively) are examined both in non-denaturing and denaturing gels; in the former we are looking for a single band of approximately the correct apparent molecular weight, while in the latter we are looking for linear strands of the lengths predicted for ligation. Ligation of double crossover molecules has previously been shown to be well-behaved (Li et al. 1996). Topologically closed structures, such as ABCD, can be assayed by treating with an exonuclease (Ma et al. 1986). Although none of these tests is absolutely rigorous, together they may give us confidence that the reactions are proceeding as predicted.

4.1.1 Materials and Methods

Sequence Design. The twelve strands required for A, B, C, D, and D' were designed by applying the principles of sequence symmetry minimization (Seeman 1990), where the design process ensures that there are no complementary regions between strands, except as desired. In short, each double crossover molecule is designed by creating sequences appropriate for two asymmetric Holliday junctions, then juxtaposing these sequences as appropriate for a four-stranded DAO, adding

hairpin sequences and re-phasing A1 and C1 to put the nick in the central region of the DAO. A1's hairpin regions are longer than C1's to allow A1 and C1 to be distinguished on gels. The lengths of the linkers B and D were chosen such that both DAO units should be nearly coplanar according to an estimated 10.5 base pairs per double helical full turn. Exact sequences are given in Figure 4.1.

Synthesis and Purification of DNA. All strands were synthesized on an Applied Biosystems 380B automatic DNA synthesizer using routine phosphoramidite procedures (Caruthers 1985). DNA strands were purified by denaturing polyacrylamide gel electrophoresis. DNA concentrations were estimated by OD_{260}. All strands were phosphorylated by T4 Polynucleotide Kinase (U.S. Biochemical or Promega), followed by phenol extraction and ethanol precipitation. DNA was not radiolabeled, with the exception of the DNA used for Figure 4.6, for which 10% of strands were phosphorylated with $^{32}\gamma$-ATP and mixed with 90% non-radiolabelled strands.

Formation of Hydrogen-Bonded Complexes. Complexes A, B, C, D, and subportions thereof were formed by mixing stoichiometric quantities of each strand at concentrations near 1 μM in 1x USB T4 DNA Ligase buffer (U.S. Biochemical: 66 mM Tris·HCl (pH 7.6), 6.6 mM MgCl₂, 10 mM DTT, 66 μM ATP; or Promega: 30 mM Tris·HCl (pH 7.8), 10 mM MgCl₂, 10 mM DTT, 500 μM ATP). These solutions were annealed for two hours from $80^\circ C$ down to room temperature.

Formation of Covalently Bonded Complexes. Complexes AB, BC, ABC, ABCD, and ABCD' were formed by mixing stoichiometric quantities of annealed A, B, C, and D', followed by D after 20 minutes. Up to 50 units of T4 DNA Ligase (U.S. Biochemical or Promega) were added and solutions were incubated in a $16^\circ C$ water bath for 2 or 8 hours. One sample of ABCD was further treated by adding $\frac{1}{10}^{th}$ volume 10x USB Exonuclease III buffer (U.S. Biochemical) and 100 units Exonuclease III (U.S. Biochemical), incubated at $37^\circ C$ for 1 hour. Prior to being loaded in gels, solutions for gels (a) and (b) were heated to $80^\circ C$ and again annealed to room temperature, to denature proteins and re-form hydrogen-bonded complexes. For gels (c), ligation was followed by phenol extraction and ethanol precipitation, then samples were heated to $90^\circ C$ for 5 minutes prior to being loaded.

Denaturing Polyacrylamide Gels. Denaturing gels contain 8.3 M urea and 8% acrylamide (19:1 acrylamide:bisacrylamide). The running buffer is TBE (89 mM Tris·HCl (pH 8.0), 89 mM boric acid, 2 mM EDTA). The sample buffer contains 0.1% bromphenol blue and xylene cyanol FF tracking dyes in 80% formamide with 10 mM EDTA. Samples are heated at $80^\circ C$ for 5 minutes immediately prior to loading. Gels are run at approximately 60 V/cm and 35 Watts, then soaked in StainsAll dye and digitized by DeskScan II on an Apple Macintosh.

Non-denaturing Polyacrylamide Gels. Non-denaturing gels contain 12.5 mM Mg^{++} and 8% acrylamide (19:1 acrylamide:bisacrylamide), 0.75 mm thick. The running buffer is TAE/Mg^{++} (40 mM Tris·HCl (pH 8.0), 20 mM acetic acid, 2 mM EDTA, 12.5 mM magnesium acetate). The loading buffer contains 0.02% bromphenol blue and xylene cyanol FF tracking dyes and 5% glycerol in ligation buffer. Gels are run at approximately 16 V/cm and 10 Watts at $4^\circ C$, then soaked in StainsAll dye and digitized by DeskScan II on an Apple Macintosh.

4.1.2 Results

Formation of Complexes.

The first question is whether the individual duplexes and double-crossover molecules A, B, C, and D will form from their component strands. Figure 4.2 shows a non-denaturing "formation gel" for double-crossover molecules A and C. Each lane contains a different subset of strands. Each lane shows a single prominent band, indicating that the strands form a specific complex. Faint bands are presumably the result of poor stoichimetry. For example, a deficit of A3 in lane 5 could

explain the minor production of `A12` in this lane. We estimate that the bands for `A` and `C` in lanes 3 and 8 contain over 90% of the material; however, this is not a quantitative measurement. The mobilities of the partial complexes for `A` are as would be expected for double-stranded DNA of the same molecular weight: `A23` < `A12` < `A123`. Lane 9 contains an anomolous major band: rather than seeing `C12` at the 132nt level, we see a major and minor band above the 200 nt level. We now interpret this as a `C12C12` dimer bound together at the self-complementary `Nde I` and `Sal I` restriction sites, as shown in Figure 4.3. In the additional presence of `C3`, these site are double stranded and the dimer is not produced. Only one of the arms of `A` has a restriction site, which explains why a `A12A12` dimer was not also observed.

7% Non-denaturing PAGE

1	2	3	4	5	6	7	8	9	10

600 nt

400 nt

200 nt

264? C12C12

172: A
164: C
140: A12
132: C12

84: A23, C23

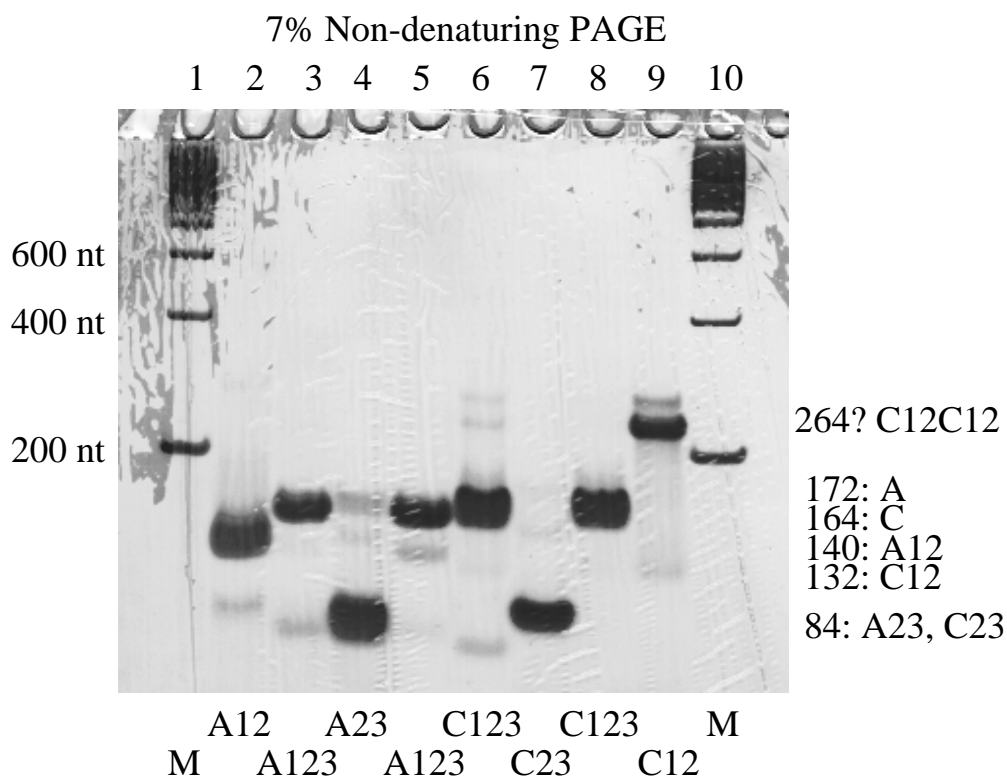A12   A23   C123   C123   M
M   A123   A123   C23   C12

Figure 4.2: Non-denaturing gel. Lanes 1 and 10: 100 base-pair double-stranded ladder. Lane 2: `A12` (140 nucleotides). Lanes 3 and 5: `A123` (172 nucleotides). Two preparations of strand `A1` were used, which apparently have different stoichiometry. Lane 4: `A23` (84 nucleotides). Lanes 6 and 8: `C123` (164 nucleotides). Two preparations of strand `C1` were used, which apparently have different stoichiometry. Lane 7: `C23` (84 nucleotides). Lane 9: `C12` (132 nucleotides). The band at ≈ 240 nt is presumably the dimer `C12C12`.

Figure 4.4 shows several stages in the formation of `ABCD`. On the non-denaturing gel, the duplexes and double crossover molecules `A`, `B`, `C`, and `D` form clean bands (a, lanes 1-4) which migrate with approximately the same mobility as equivalent molecular weight duplex DNA. Ligation products `AB` and `BC` also show clean bands (a, lanes 9-10). Ligation product `ABC` appears as the major band in its lane (a, lane 8); another band appears at the level of `AB` and `BC` indicating incomplete ligation. Ligation product `ABCD` also appears, we believe, as the major band in its lane (a, lanes 7 and 5); a slower unidentified band also appears. After exonuclease treatment, the major
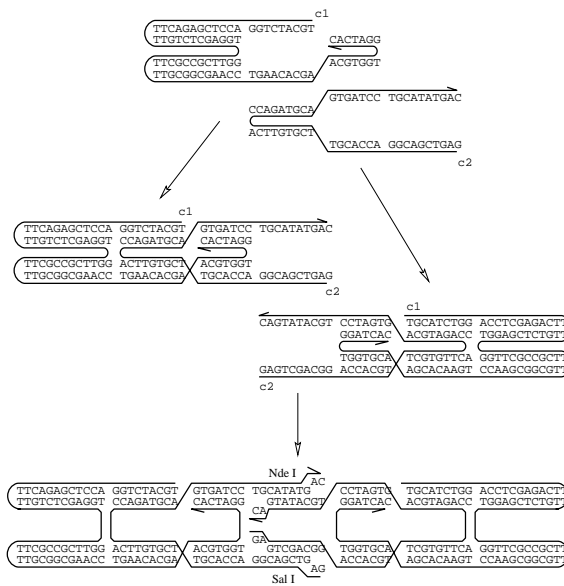
CACTAGG
TTCAGAGCTCCA GGTCTACGT c1
TTGTCTCGAGGT
ACGTGGT
TTCGCCGCTTGG
TTGCGGCGAACC TGAACACGA

CCAGATGCA GTGATCC TGCATATGAC
ACTTGTGCT
TGCACCA GGCAGCTGAG c2

TTCAGAGCTCCA GGTCTACGT GTGATCC TGCATATGAC
c1
TTGTCTCGAGGT CCAGATGCA CACTAGG
TTCGCCGCTTGG ACTTGTGCT ACGTGGT
TTGCGGCGAACC TGAACACGA TGCACCA GGCAGCTGAG
c2

CAGTATACGT CCTAGTG TGCATCTGG ACCTCGAGACTT c1
GGATCAC ACGTAGACC TGGAGCTCGTT
TGGTGCA TCGTGTTCA GGTTCGCCGCTT
GAGTCGACGG ACCACGT AGCACAAGT CCAAGCGGCGTT
c2

Nde I
AC
TTCAGAGCTCCA GGTCTACGT GTGATCC TGCATATG CCTAGTG TGCATCTGG ACCTCGAGACTT
TTGTCTCGAGGT CCAGATGCA CACTAGG GTATACGT GGATCAC ACGTAGACC TGGAGCTCGTT
CA
TTCGCCGCTTGG ACTTGTGCT ACGTGGT GTCGACGG TGGTGCA TCGTGTTCA GGTTCGCCGCTT
TTGCGGCGAACC TGAACACGA TGCACCA GGCAGCTG ACCACGT AGCACAAGT CCAAGCGGCGTT
GA
AG
Sal I

Figure 4.3: Proposed structure causing the anomolous band in Figure 4.2 lane 9. First C1 and C2 hybridize to form C12, then two C12 molecules bind at self-complementary restriction enzyme sites.

band of ligation product ABCD is still apparent, though diminished (a, lane 6).

On the denaturing gel, we obtain further evidence of ligation activity by observing the lengths of newly created oligonucleotides. Lanes 1-4 can be used as markers for the lengths of most of the original oligonucleotides: A1 (88), A2 (52), B1 (64), B2 (52), C1 (80), C2 (52), D1 (64), D2 (52). A3 and C3, both 32 nucleotides, ran off the gel. Lane 10, product AB, shows the expected formation of A2B1 (116) and B2A3 (84); lane 9, product BC, likewise shows the formation of B1C2 (116) and C3B2 (84); and lane 8, product ABC, shows the expected formation of A2B1C2 (168) and C3B2A3 (116). Lanes 5 and 7, product ABCD, contain only three significant bands: A1 (88), C1 (80), and a band which migrates slower than a 2000 nucleotide strand, according to the marker (lane 11). This slow band is exonuclease-resistant (lane 6). We therefore conclude that the band contains the catenane A2B1C2D1:A3D2C3B2; i.e., ABCD minus the A1 and C1 loops, which apparently were not ligated. Double crossover molecules with two nicks have been shown to be stable (Zhang and Seeman 1994a), suggesting that the nicks in A1 and C1 should not significantly affect the formation or stability of A or C.

We wished further confirmation of the idenity of the bands. It is possible to determine exactly which set of oligonucleotides is present in each band by "differential labelling." Here, we run a set of nearly identical experiments, the only difference being which oligonucleotide has been phosphorylated with ^{32}P. On the gel, only products incorporating the radiolabelled oligonucleotide will be visisble. Thus, for each band, we can simply read off the lanes in which it appears to determine which oligonucleotides are involved in the complex. If more than one complex co-migrate, the analysis gets more difficult. This is indeed what happens in the gels shown in Figure 4.5: on the 8% gel, the outer circle and the outer:inner complex comigrate. Change in mobility with change in polyacrylamide density distinguishes different topological species, allowing us to separate the outer circle and outer:inner complex on a 5% gel. However, at this percentage, the inner circle
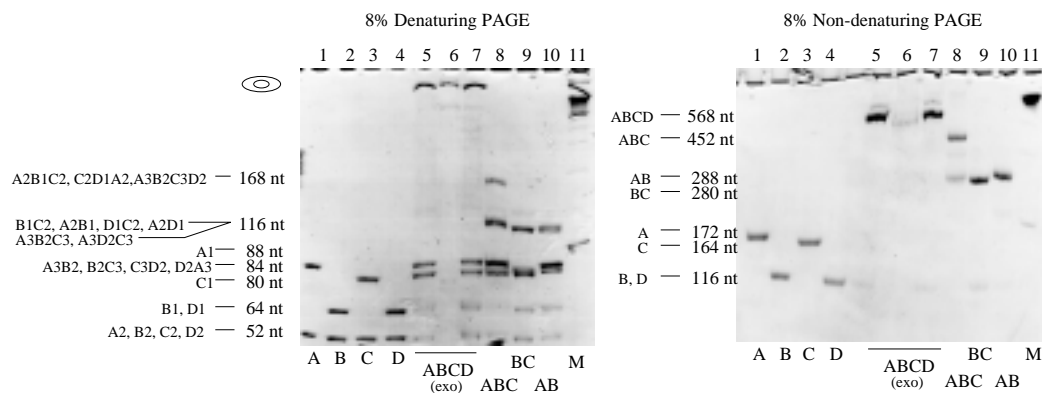
Figure 4.4: (a) Denaturing gel electrophoresis. Lane 1: A (172 nucleotides). Lane 2: B (116 nucleotides). Lane 3: C (164 nucleotides). Lane 4: D (116 nucleotides). Lane 5: Product ABCD with ligase. ABCD contains 568 nucleotides. Lane 6: Product ABCD with ligase and exonuclease. Lane 7: product ABCD with ligase. Lane 8: Product ABC with ligase. ABC contains 452 nucleotides. Lane 9: Product BC with ligase. BC contains 280 nucleotides. Lane 10: Product AB with ligase. AB contains 288 nucleotides. Lane 11: 100 base-pair double-stranded ladder. Numbers at right indicate estimated positions for expected products, consistent with the marker lane. (b) Non-denaturing gel electrophoresis: Lane 1: A (172 nucleotides). Lane 2: B (116 nucleotides). Lane 3: C (164 nucleotides). Lane 4: D (116 nucleotides). Lane 5: Product ABCD with ligase. ABCD contains 568 nucleotides. Lane 6: Product ABCD with ligase and exonuclease. Lane 7: product ABCD with ligase. Lane 8: Product ABC with ligase. ABC contains 452 nucleotides. Lane 9: Product BC with ligase. BC contains 280 nucleotides. Lane 10: Product AB with ligase. AB contains 288 nucleotides. Lane 11: 100 base-pair double-stranded ladder. Numbers at right indicate estimated positions for expected products, consistent with the marker lane.

and the 232-mer linear strand comigrate! Note that ligase was not as efficient in this experiment as it was in the previous experiment: the ABCD lane contains many partial products, indicating incomplete ligation of nicks in the ABCD complex, or poor stoichiometry so that many ABCD complexes were lacking one or more strand. This is turns out to be convenient for interpretting the next experiment, where ligation was again incomplete.

Specificity of Reaction.

Figure 4.6 shows the results of a preliminary experiment investigating the effectiveness of D vs D' in filling the slot created by ABC. Lane 13 contains ABC, and thus has primary bands for A2B1C2 (168) and C3B2A3 (116). Lanes 2 and 12 show ligation of ABCD in 1:1:1:1 stoichiometry. The ligation apparently was not as complete as in 4.4, as several bands of "partial products" are observed. The fastest band is appropriate for linear A3D2C3B2 (168) and cyclical permutations; the next band is appropriate for linear B1C2D1 or D1A2B1 (180); the next major band is appropriate for A2B1C2D1 (232) and cyclical permutations. The band below **c** is known from other gels (not shown) to be exonuclease-resistant, and the two cyclic molecule bands are thought to be an indicator of the formation of ABCD. Lanes 1 and 10 show ligation of ABC with respectively an equimolar amount or a 20-fold excess of D'. We again see the linear bands of lengths 168, 180, and 232, while bands at 136 (D2C3B2) and 116 (C3B2A3 and A3D2C3) become significant. Critically, the slow circular products are missing, suggesting that D' was only ligated on the side where it
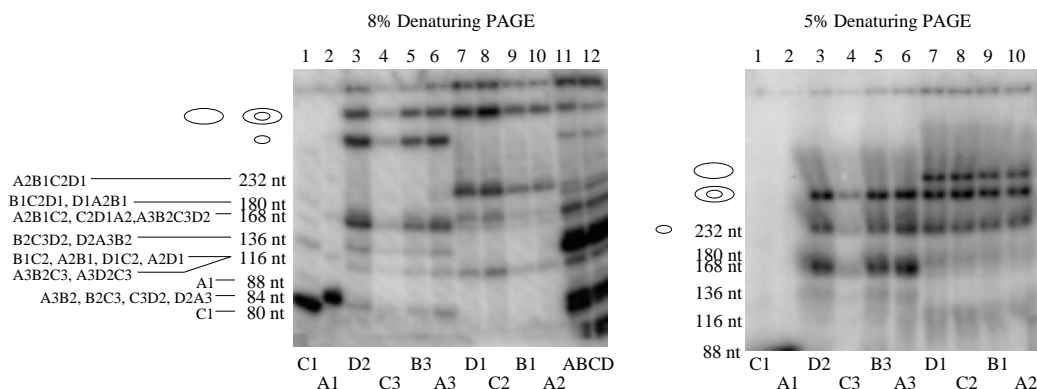
Figure 4.5: Denaturing gel electrophoresis with radiolabelled strands. All lanes contain ABCD, but in each lane only one strand was radiolabelled. (Note that the gels were improperly dried, leading to "smearing" after the gel was run.)
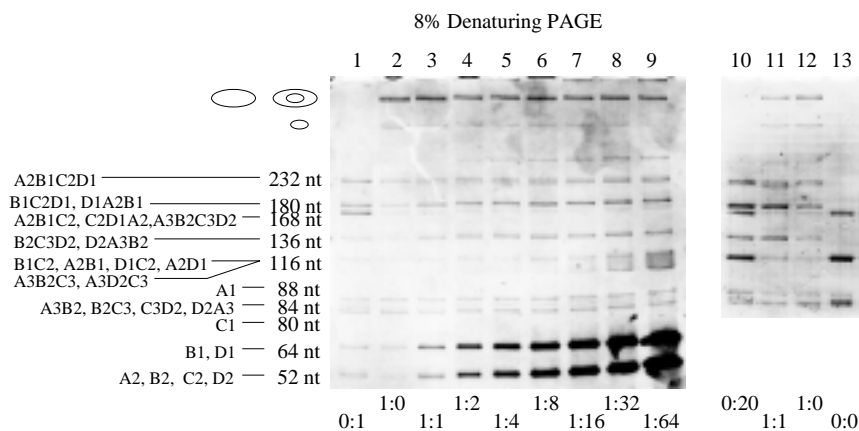

Figure 4.6: Denaturing gel electrophoresis. All lanes contain A:B:C in 1:1:1 stoichiometry, plus various concentrations of D:D'. Lane 1–9: D:D' = 0:1, 1:0, 1:1, 1:2, 1:4, 1:8, 1:16, 1:32, 1:64. Lanes 10–13: D:D' = 0:20, 1:1, 1:0, 0:0. (Note: the first gel was slightly ripped during staining.)

matches the slot's sticky end. Lanes 11 and 3–9 show ligation of one unit of ABC with an x-fold excess of D' and equimolar D, where x ranges from 1 to 64. In every case, the closed molecule **c** is formed, indicating that ABCD is still formed in the presence of competing D'. Additional bands also appear, possibly due to unexpected interactions involving D1 or D2.

4.1.3 Discussion

We interpret these results as follows. First, we believe that we are making the ABCD complex, with the caveat that we believe the nicks in A1 and C1 are not being sealed. This suggests that (1) in ABC, the linker B is properly spaced such that double crossover molecules A and C are roughly coplanar, and (2) that in each double crossover molecule, the two helical axes are also roughly coplanar. Second, we observe that D, which matches the sticky ends on both sides of ABC's slot,

out-competes D', which matches only on one side, even when D' is 64-fold more abundant than D. We plan to quantitate the thermodynamics of this reaction in the near future.

The experiments reported here bear on the two-dimensional self-assembly process postulated in Section 3.2.5. These experiments are meant to model a single slot-filling step during the self-assembly of a two dimensional lattice. In our experiments, this fundamental step can occur, and with some specificity. These results encourage us to examine this step in closer detail in the future, as well as to attempt the self-assembly of an entire sheet. We hope that the self-assembly of an algorithmically patterned sheet of DNA can eventually be verified by TEM or AFM microscopy.

The self-assembly of molecules can correspond to several well known computational classes up to and including universal computation. This suggests that external processing is not an intrinsic element of molecular computation; computationally universal "one-pot" reactions seem plausible. We have shown some encouraging but preliminary experimental investigations into the fundamental computational step in our two dimensional self-assembly model. The generality of the approach used here suggests that the potential for universal computation may be widespread among self-assembly processes in nature. In addition to being interesting in its own right as a universal mechanism, it may be worth considering whether the self-assembly processes described here could be useful technologically, perhaps as part of an approach to nanotechnology (Li et al. 1996).

4.2   Experiments with 2D Lattices

Abstract[2] Molecular self-assembly presents a bottom-up approach to the fabrication of objects specified with nanometer precision. DNA molecular structures and intermolecular interactions are particularly amenable to design and are sufficient for the creation of complex molecular objects. Here we report the design and observation of two-dimensional crystalline forms of DNA that self-assemble from synthetic DNA double-crossover molecules. Intermolecular interactions are programmed by the design of sticky ends that associate according to Watson-Crick complementarity, enabling us to create specific periodic patterns on the nanometer scale. The patterned crystals have been visualized by atomic force microscopy.

Control of the detailed structure of matter on the finest possible scale is a major goal of chemistry, materials science and nanotechnology. This goal may be approached in two steps, (1) the construction of individual molecules, represented by the triumphs of synthetic chemistry, and (2) the arrangement of molecular building blocks into larger structures. The simplest arrangement of molecular units, in two or three dimensions, is a crystal. Design components for crystals must have definable intermolecular interactions and must be rigid enough to prevent the formation of ill-defined aggregates (Liu et al. 1994). Branched DNA molecules with sticky ends appear to be

promising for macromolecular crystal design (Seeman 1982), because their intermolecular interactions can be programmed through sticky ends (Cohen et al. 1973) that associate to form B-DNA (Qiu et al. 1997); however, studies of three- and four-arm junctions reveal that the angles flanking their branchpoints are flexible (Ma et al. 1986; Petrillo et al. 1988).

The need for a rigid design component with predictable and controllable interactions has led to the utilization of the antiparallel DNA double-crossover motif (Fu and Seeman 1993) for this purpose. Double-crossover (DX) molecules are analogs of intermediates in meiosis (Schwacha and Kleckner 1995) which consist of two side-by-side double-stranded helices linked at two crossover junctions. Antiparallel DX molecules have been shown to possess the rigidity lacking in conventional branched junctions, thus suggesting that they might be suitable for use in the assembly of periodic matter (Li et al. 1996).

These findings stimulated a theoretical proposal to use two-dimensional (2-D) lattices of DX molecules (Winfree 1996b) for DNA-based computation (Adleman 1994). In the mathematical theory of tilings (Grünbaum and Shephard 1986), rectangular tiles with programmable interactions, known as Wang tiles, can be designed so that their assembly must mimic the operation of a chosen Turing Machine (Wang 1963). DX molecules acting as molecular Wang tiles could self-assemble to perform desired computations (Winfree 1996b; Winfree et al. in press; Reif in press). Consequently, the ability to create 2-D lattices of DX molecules assumes additional interest as a step toward the design of molecular algorithms.

Here, we report the assembly from DX molecules of three 2-D lattices with two distinct topologies. The DX molecules, $\sim 2 \times 4 \times 13$ in size, self-assemble in solution to form single-domain crystals as large as 2×8 microns with uniform thickness between 1 and 2 nm, as visualized by atomic force microscopy (Binnig et al. 1986) (AFM). By incorporating a DNA hairpin into a DX molecule to serve as a topographic label, we have produced stripes above the surface at intervals of 25. Two-component lattices have been assembled with a stripe every other unit.

4.2.1 Design of DNA Crystal

Our approach to two-dimensional crystal design is derived from the mathematical theory of tiling (Grünbaum and Shephard 1986; Winfree 1996b). The desired lattice is specified by a set of Wang tiles with colored edges; the Wang tiles may be placed next to each other only if their edges are identically colored where they touch (Figure 4.7a). Our goal is to design synthetic molecular units corresponding to these tiles, such that they will self-assemble into a crystal that obeys the coloring conditions. As an initial demonstration of molecular Wang tiles, we have chosen the simplest non-trivial set of tiles: two tiles, **A** and **B**, which make a striped lattice (Figure 4.7a, left). Translated into molecular terms, we obtain DX systems that self-assemble in solution into two-dimensional crystals with a well-defined subunit structure.

The antiparallel DX motif (Fu and Seeman 1993) consists of two juxtaposed immobile 4-arm junctions (Seeman 1982) arranged such that at each junction the non-crossover strands are antiparallel to each other. There are five distinct DX motifs, but only two are stable in small molecules (Fu and Seeman 1993): these are called DAO (double crossover, antiparallel, odd spacing) and DAE (double crossover, antiparallel, even spacing). The design depends critically upon the twist of the B-form DNA double helix, in which a full turn takes place in ~ 10.5 base pairs (Wang 1979; Rhodes and Klug 1980). DAO molecules have an odd number of half-turns (e.g. 3 half-turns is ~ 16 base pairs) between the crossover points, while DAE molecules have an even number of half-turns (e.g. 4 half-turns is ~ 21 base pairs). Computer models of the DX molecules used in this study, shown in Figure 4.7b, were generated using NAMOT2 (Carter and Tung 1996). Complete base stacking at the crossover points is assumed. The DAO molecules consist of 4 strands
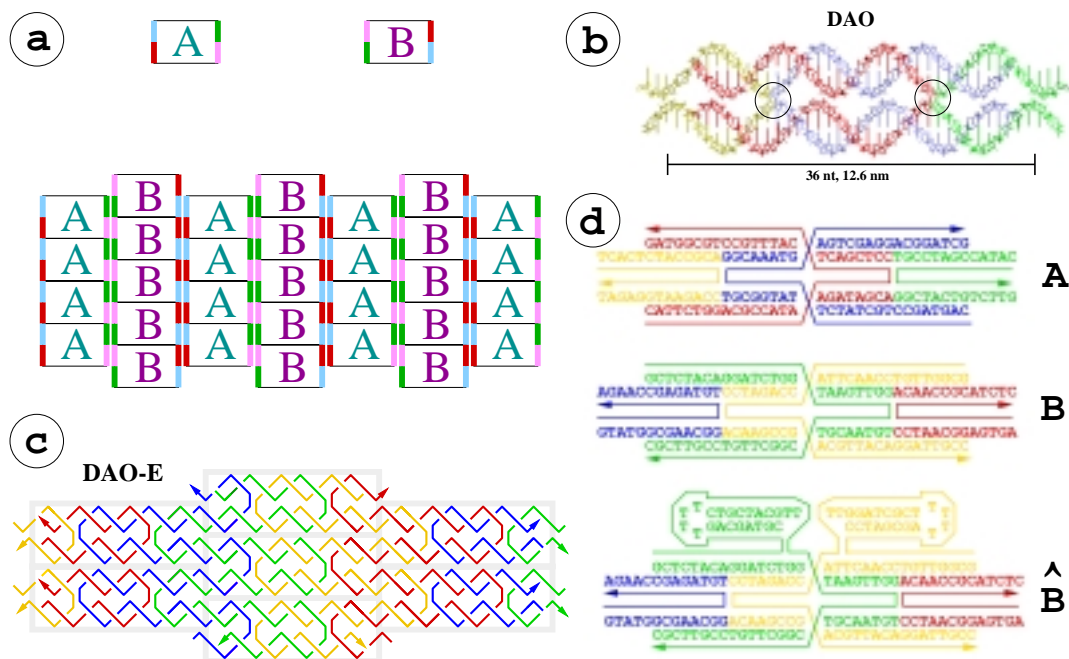
Figure 4.7: Design of DX molecular structure and arrangement into 2-D lattices. **a**, The logical structure for 2-D lattices consisting of two units. Type **A** units have four colored edge regions, each of which match exactly one colored region of the adjacent type **B** units. Note that rotations and reflections of Wang tiles are disallowed; an equivalent restriction could also be obtained by using non-rectangular tiles or more complex patterns of colors. **b**, Model structures for DAO type **A** units. Each component oligonucleotide is shown in a unique color. The crossover points are circled. **c**, The lattice topologies produced by the DAO. Each DX unit is highlighted by a grey rectangle. A unique color is chosen for each strand type which would be formed after covalent ligation of units. Arrowheads indicate the 3′ ends of strands. Black ellipses indicate dyad symmetry axes perpendicular to the plane; black arrows indicate dyad axes in the plane (full arrowhead) or screw axes (half arrowhead). **d**, The actual sequences used in the reported experiments (see Methods for several exceptions). The schematics accurately report intended primary and secondary structure – oligonucleotide sequence and paired bases – but are not geometrically or topologically faithful because they don't show the double helical twist. Both type **B** and type**B̂** are shown, indicating where the hairpin sequences are inserted.

of DNA, each of which participates in both helices. The DAE molecules consist of 3 strands that participate in both helices (yellow, light blue, green), and 2 strands that do not cross over (red, dark blue). Each corner of each DX unit has a single-stranded sticky end with a unique sequence; specific association of DX units is controlled by choosing sticky ends with Watson-Crick complementarity.

To ensure that the component strands form the desired complexes, strand sequences must be designed carefully so that alternative associations and conformations are unlikely. Therefore we must solve the "negative design problem" (Seeman 1990; Yue and Dill 1992; Sun et al. 1996) for DNA: find sequences that maximize the free energy *difference* between the desired conformation and all other possible conformations. We use the heuristic principle of sequence symmetry

minimization (Seeman 1982, 1990) to minimize the length and number of unintentional Watson-Crick complementary subsequences. In each DX molecule's sequences, there are no 6-base subsequences complementary to other 6-base subsequences except as required by the design, and spurious 5-base complementarity is rare. Thus it is expected that during self-assembly, the DNA strands spend little time in undesired associations and form DX units with high yield.

DX units can be designed that will fit together into a two-dimensional crystalline lattice. Here we use two distinct DAO unit types (Figure 4.7a) to produce striped lattices. The lattices produced by this system is called DAO-E to indicate the number of half-turns between crossover points on adjacent units; its topology is shown in Figure 4.7c. The symmetries of the DAO-E lattice is that corresponding to the layer group (Vainshtein 1994) $p2_122$. Covalently joining adjacent nucleotides at nicks in the lattice, by chemical or enzymatic ligation, would result in a "woven fabric" of DNA strands. Ligation of the DAO-E design produces four distinct strand types, each of which continues infinitely in the vertical direction (in this paper, "vertical" and "horizontal" will always be as in Figure 4.7c).

Control of self-assembly to yield the 2-D lattice is obtained by two design criteria. First, the sticky-end sequences for each desired contact are unique; this ensures that the orientations and adjacency relations of the DX units comply exactly with the design in Figure 4.7a. Sticky ends are length 5, so that each correct contact contributes approximately 8 to the free energy of association at 25°C, according to a nearest-neighbor model (SantaLucia et al. 1996). Second, the lengths of the DX arms and sticky ends, and thus the separations between crossover points, respect as closely as possible the natural twist of the B-form DNA double helix; thus adjacent DX molecules are effectively coupled by torsional springs whose equilibrium positions have been designed to keep the adjacent DX molecules coplanar. For example, a linear rather than planar polymer could result if each unit makes two sticky-end bonds to each neighboring unit, but this would require overtwisting, undertwisting, or bending of the double helix, and thus is discouraged by our design.

Figure 4.7d shows the DX units and sequences used in our experiments, except as noted in Methods. In each system, there are two fundamental DX units, called **A** and **B**, and, additionally, an alternative form **B̂** that contains two hairpin-terminated bulged 3-arm junctions (similar to the DX+J motif (Li et al. 1996)). Based on studies of bulged 3-arm junctions (Ouporov and Leontis 1995), we expect that in each unit, one hairpin will point up and out of the plane of the DX crystal, while the other hairpin will point down and into the plane, without significantly affecting the rigidity of the molecule (Li et al. 1996). The **B̂** units will replace the **B** units and serve as contrast agents for AFM imaging, because their increased height can be measured directly.

4.2.2 Materials and Methods

DNA sequences and synthesis. Figure 4.7d shows sequences used in these experiments; for historical reasons, some figures show experiments where variants of these sequences were used. The sequences for DAO-E **A**, **B**, and **B̂** given in Figure 4.7d were used for Figure 4.12bc. Figure 4.12adef show DAO systems with symmetrical sticky ends: the sequence for the green strands of DAO **A** is 5′TCACT...GAGAT3′ and the sequence for the blue strands of DAO **B** and **B̂** are 5′AGTGA...ATCTC3′. All oligonucleotides were synthesized by standard methods, PAGE purified, and quantitated by UV absorption at 260 nm in H_2O.

Annealing of oligonucleotides. The strands of each DX unit were mixed stoichiometrically and dissolved to concentrations of .2 to 2 μM in TAE/Mg^{++} buffer (40 mM Tris·HCl (pH 8.0), 1 mM EDTA, 3 mM Na$^+$, 12.5 mM Mg^{++}). The solutions were annealed from 90°C to room temperature over the course of several hours in a Perkin-Elmer PCR machine (to prevent concentration by evaporation). To produce lattices, equal amounts of each DX were mixed and annealed

from $50^\circ C$ to $20^\circ C$ over the course of up to 36 hours. In some cases (Figure 4.12abc) all strands were mixed together from the very beginning.

Gel electrophoresis studies. For gel-based studies, T4 polynucleotide kinase (Amersham) was used to phosphorylate strands with ^{32}P; these strands were then PAGE purified and mixed with an excess of unlabeled strands. Non-denaturing 4%, 5%, or 8% PAGE (19:1 acrylamide:bis-acrylamide) in TAE/Mg^{++} was performed at 4°C or at room temperature. For denaturing experiments, after annealing in T4 DNA ligase buffer (Amersham) (66 mM Tris·HCl(pH 7.6), 6.6 mM MgCl$_2$, 10 mM DTT, 66 μM ATP), 1 μl = 10 units T4 DNA ligase (Amersham) was added to 10 μl DNA solution and incubated for up to 24 hours at 16°C or at room temperature. For exonuclease reactions, 50 units of exonuclease III (Amersham) and 5 units of exonuclease I (Amersham) were added after ligation, and incubated an additional 3.5 hours at 37°C. The solution was added to an excess of denaturing dye buffer (0.1% xylene cyanol FF tracking dye in 90% formamide with 1 mM EDTA, 10 mM NaOH) and heated to 90°C for at least 5 minutes prior to loading. Denaturing gels contained 4% acrylamide (90:1 acrylamide:bisacrylamide) and 8.3 M urea in TBE (89 mM Tris·HCl (pH 8.0), 89 mM boric acid, 2 mM EDTA). Gels were analyzed by phosphorimager.

Preparation of AFM sample. 2 to 10 μl were spotted on freshly cleaved mica (Ted Pella, Inc) and left to adsorb to the surface for 2 minutes. To remove buffer salts, 5 to 10 drops of doubly-distilled or nanopure H$_2$O were placed on the mica, the drop was shaken off and the sample was dried with compressed air. Imaging was performed under isopropanol in a fluid cell on a NanoScope II using the D or E scanner and commercial 200 μm cantilevers with Si$_3$N$_4$ tips (Digital Instruments). The feedback setpoint was adjusted frequently to minimize contact force to approximately 1 to 5 nN. Images were processed with a first- or third-order "flatten filter," which independently subtracts a first- or third-order polynomial fit from each scanline to remove tip artifacts; however, this technique introduces false "shadows" into the images shown here.

4.2.3   Results of Characterization by Gel Electrophoresis

A prerequisite for lattice self-assembly is the formation of the DX units from their component strands. A thorough investigation of this issue was done for the original studies of DX (Fu and Seeman 1993); that the new designs also behave well constitutes further validation of the antiparallel DX motif. Because the sticky ends of **A** units have affinity only for sticky ends of **B** units, and not for themselves, neither **A** nor **B** alone in solution can assemble into a lattice. Thus the formation of isolated DX units can be monitored easily by non-denaturing gel electrophoresis, as described previously (Fu and Seeman 1993), and greater than 95% of the material is seen in the expected band for **A** and greater than 85% for **B**(Figure 4.8). Additionally, complexes are formed *only* by strands which were designed to interact. However, strand 3 of **B** does not bind fully to strand 4, unless strand 2 is also present. This may be due to a potential hairpin structure near the $5'$ end of strand 3; when strand 2 binds to strand 3, we postulate that this hairpin is undone. Note also that dimer and multimer species are not found, and in particular note that the individual **A** or **B** monomers do not assemble into extended structures, such as the desired lattice.

Solutions containing **A** units and **B** units can be mixed and annealed to form **AB** lattices. Lane 4 of Figure 4.9(left) shows that the self-assembled structure is too large to migrate through the gel, although a fraction of the material is coming out of the well in a smear. Enzymatic ligation of these lattices with T4 DNA ligase produces immobile material, while the **A** and **B** units alone are not substrates for ligation (Figure 4.9(left)). The nicks in the lattice, where strands from adjacent DX units abut, are all on the upper or lower surface of the lattice, where they are accessible to the enzyme. The ligated lattice should contain long covalent DNA strands, which serve as reporters of successful lattice formation (shorter strands report either the presence of
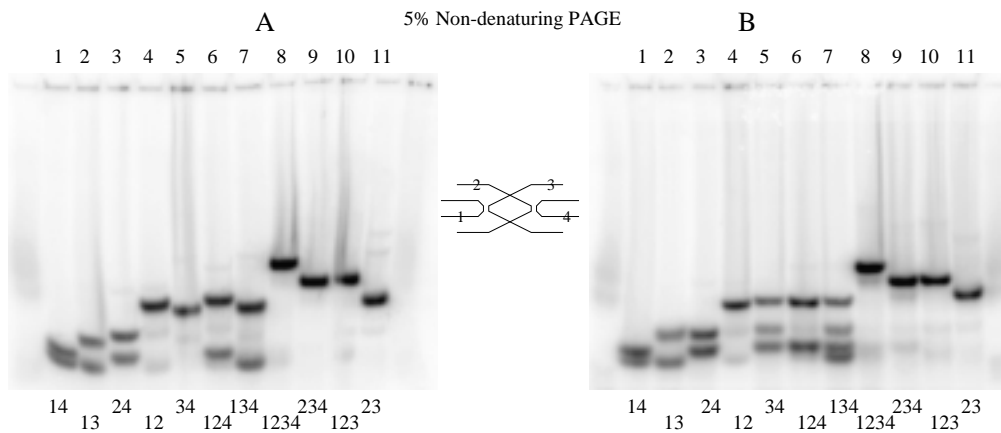
Figure 4.8: Formation gels for the **A** and **B** double-crossover molecules, run at $l^\circ C$ by 5% non-denaturing PAGE. Every strand is radiolabelled so that quantitation is possible.

small aggregates or an occasional failure to ligate within the lattice). All four reporter strands extend for more than 30 repeats when visualized by denaturing polyacrylamide gel electrophoresis (Figure 4.9, right). Longer strands co-migrate on this gel, so we cannot determine the full extent of polymerization. These results suggest that the lattice is a good substrate for T4 DNA ligase, and that the lattices can form with more than 30×30 units. However, unintended associations or side reactions could lead to similar distributions of strand lengths after ligation. Direct physical observation is necessary to confirm lattice assembly.
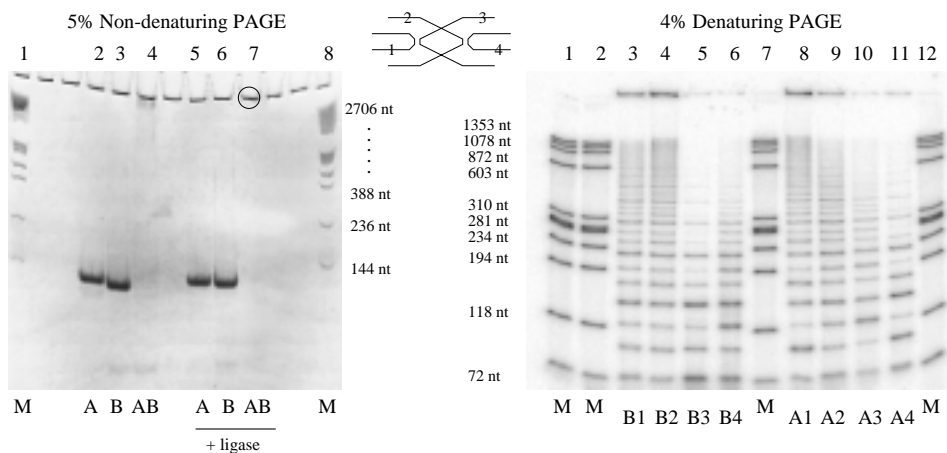

Figure 4.9: (left) Non-denaturing gel, StainsAll stain. Lanes 2-4 show **A** alone, **B** alone, and **AB** together. Lanes 5-7 show **A** alone ligated, **B** alone ligated and **AB** together ligated. All the material in lane 7 is in a sharp band directly around the well (circled); this band can be seen clearly in color, but not in the B/W rendition here. (right) Denaturing gel, with differential radiolabelling. Every lane contains either marker or ligated **AB**; in each lane the radiolabelled oligonucleotide is indicated below.

As a control to test the 4-connected nature of the putative lattice product, versions of **A** and **B** were made with certain sticky ends truncated, as shown in Figure 4.10. $\mathbf{A}_v$ and $\mathbf{B}_v$ were designed to make vertical one-dimensional chains of DX units; $\mathbf{A}_h$ and $\mathbf{B}_h$ were designed to make horizontal one-dimensional chains of DX units; and $\mathbf{A}_d$ and $\mathbf{B}_d$ were designed to make diagonal one-dimensional chains of DX units. However, quite unlike the **AB** product stuck in the well, all three truncated systems produced what appear to be dimers on the non-denaturing gel. No ligase was used. Apparently, the chain structures are either not being made, or they are falling apart into dimers in the gel. We do not yet understand these results.
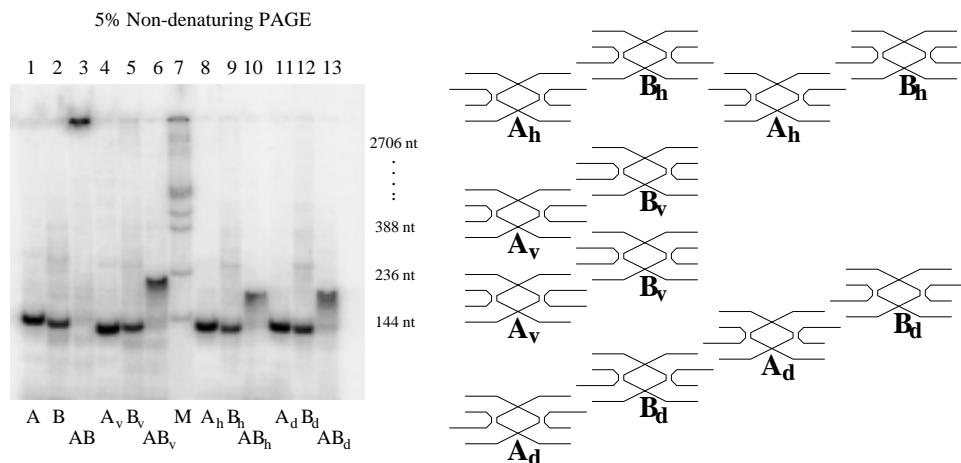

Figure 4.10: (left) 5% Non-denaturing gel, with radiolabelling of **A**3 and **B**3. (right) Diagram of the truncated DX units, and intended chain structures.

4.2.4 Results of AFM Imaging

We have used atomic force microscopy (Binnig et al. 1986) to demonstrate unequivocally the formation of 2-D lattices. **A** and **B** units are annealed separately, then combined and annealed together to form **AB** lattices. The resulting solution is deposited for adsorption on an atomically flat mica surface, and then imaged under isopropanol by contact mode AFM (Hansma et al. 1992). The solution is not treated with DNA ligase, and thus the lattices are held together only by noncovalent interactions (e.g. hydrogen bonds and base stacking). This protocol ensures that the solution contains no protein contaminants and demonstrates that ligase activity is not necessary for the self-assembly process. Negative controls of buffer alone and of **A** or **B** alone show no aggregates larger than 20 nm (Figure 4.11abc). In separate experiments, **A** and **B** DAO units were modified by the removal of two sticky ends from each unit (e.g. all yellow and red sticky ends in Figure 4.7d); when the modified **A** and **B** units were annealed together, we observed only linear and branched structures with apparent widths typically less than 10 nm (Figure 4.11def), providing additional negative controls. However, the unmodified **AB** samples contain 2-D sheets many microns long, often more than 200 nm wide (Figure 4.12a). The apparent height of the sheets is $1.4 \pm .5$ nm, suggesting a monolayer of DNA. The sheets often seem ripped and appear to have a grain, in that rips have a preferred direction consistent with the design (Figure 4.7c). In the DAO-E lattice, a vertical rip requires breaking six sticky-end bonds per 12 nm torn, whereas a horizontal rip requires breaking only one sticky-end bond per 13 nm torn. A possible vertical

column, perpendicular to the rips, is indicated in Figure 4.12a (arrows). Although in this image the columns are barely perceptible, Fourier analysis shows a peak at 13 ± 1 nm, suggesting that observed columns are 1 DX wide. Periodic topographic features would not be expected in the ideal $\mathbf{AB}$ lattice; however a vertically stretched lattice may have gaps between the DX units that could produce the periodic features seen here. Because crystals are found in AFM samples taken from both the top and the bottom of the solution, we believe that crystals form in solution and are not due to interaction with a surface.
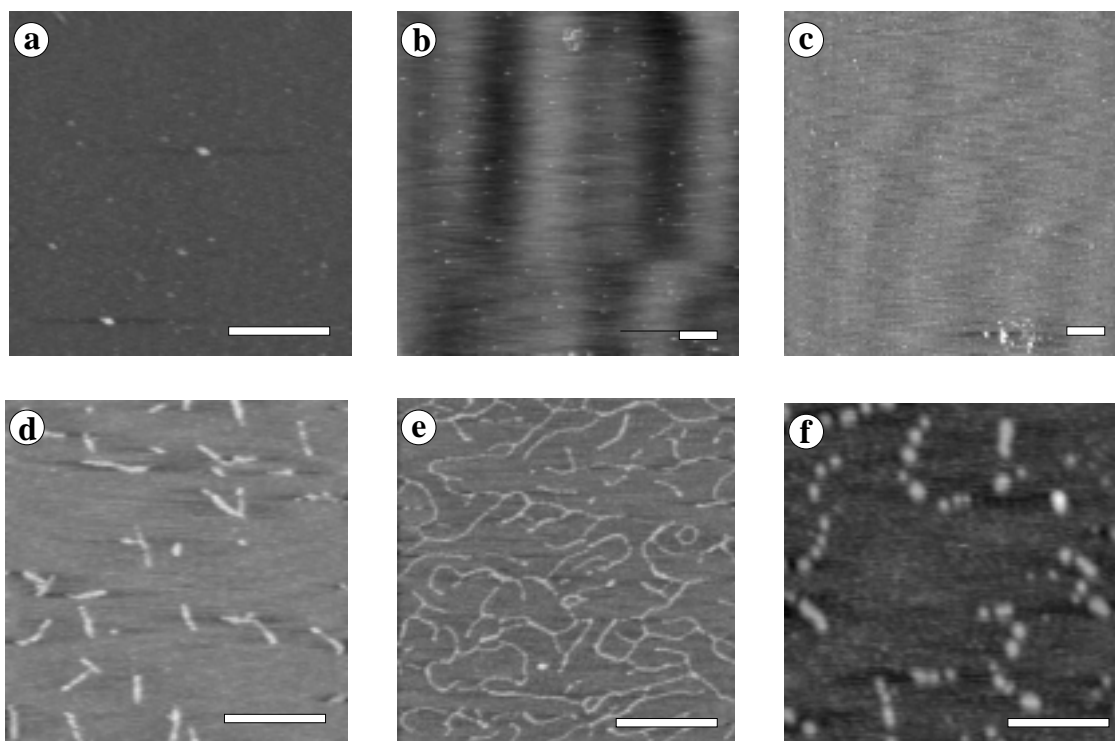

Figure 4.11: AFM images of buffer (a), $\mathbf{A}$ and $\mathbf{B}$ controls (b and c respectively), and sticky-end-truncation controls $\mathbf{AB}_v$ (d), $\mathbf{AB}_h$ (e), and $\mathbf{AB}_d$ (f). All scale bars are 300 nm; images show 500×500 nm, 1×1 μm, or 3×3 μm. The grayscale indicates height above the mica surface; apparent height of features is less than 5 nm.

4.2.5 Control of Surface Topography

The self-assembling $\mathbf{AB}$ lattice can serve as scaffolding for other molecular structures. We have decorated $\mathbf{B}$ with two DNA hairpin sequences inserted into its component strands, which we call $\hat{\mathbf{B}}$ (Figure 4.7d). So decorated, the vertical columns of the lattice become strikingly apparent as stripes in AFM images (Figure 4.12bc), further confirming the proper self-assembly of the 2-D lattice. The spacing of the decorated columns is 25 ± 2 nm for the DAO-E lattice, indicating that every other column is decorated, in accord with the design. Slow annealing at 20 C and gentle handling of the DAO-E sample during deposition and washing has produced single crystals measuring up to 2×8 μm(Figure 4.12def). Close examination shows that the stripes are continuous across the crystal, and thus it appears to be a single domain containing over 500,000 DX units.
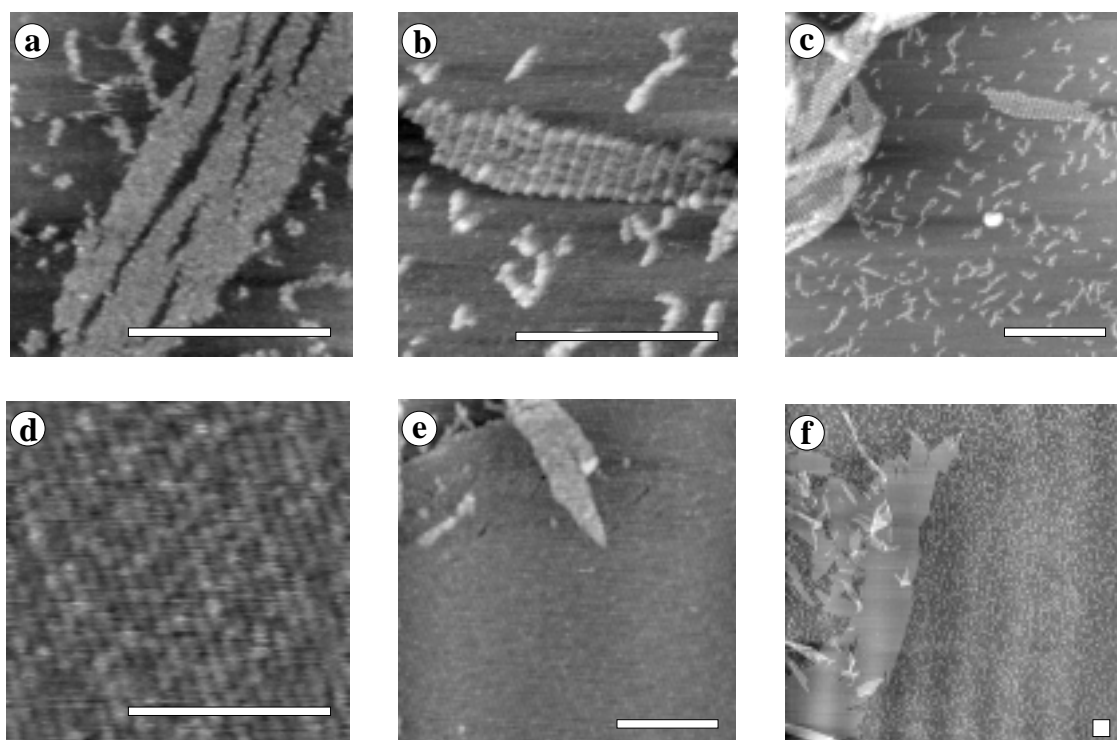
Figure 4.12: AFM images of unmodified DAO-E **AB** lattice (a). A possible vertical column is indicated by the arrows. Fourier analysis shows 13 ± 1 nm periodicity; each DAO is 12.6 nm wide. (b) and (c) show DAO-E **AB̂** lattice (two views of the same sample). Stripes have 25 ± 2 nm periodicity; the expected value is 25.2 nm. (def) show a large single-domain crystal of DAO-E **AB̂** lattice at three levels of detail (all the same sample). The largest domain is roughly 2×8 μm, and contains roughly 500,000 DX units. All scale bars are 300 nm; images show 500×500 nm, 1.5×1.5 μm, or 10×10 μm. The grayscale indicates height above the mica surface; apparent lattice height is between 1 and 2 nm.

We have also tested DAO systems incorporating only one of the two hairpins in **B̂**, DAO systems in which the 3-arm junctions are relocated by two nucleotides toward the center of the molecule. All systems produced results similar to those shown in Figure 4.12 when imaged by AFM (data not shown). The lattice assembly appears to be robust to variations in the local DX structure and is not sensitive to small variations in the annealing protocol. (Also see Winfree et al. (1998) for similar results obtained in another laboratory using different buffers, annealing conditions, and AFM instruments.)

In all images of **AB** and **AB̂** systems, we observed many DNA structures in addition to the isolated 2-D crystals discussed above. In many images the 2-D crystals appear to overlap, leading to discrete steps in thickness (Figure 4.12cde). The arrangement of crystals on the mica – solitary, overlapping, piled up like driftwood, ripped to shreds – depends sensitively upon DNA concentration and upon the sample preparation procedure, especially the wash step. Prominently, the background of every image contains small objects, which we assume to be associations of small numbers of DX units. Also, long, thin "rods" appear in some preparations (data not shown).

These structures have not been characterized.

4.2.6   Applications

The programmability of the system described here has already been demonstrated by Winfree et al. (1998), where a system of four tiles is developed to produce stripes of twice the periodicity. The produced lattice can serve as a molecular scaffold. Instead of DNA hairpins, other chemical groups can be used to label the DX molecules. Previous groups have used biotin-streptavidin-gold to label linear DNA for imaging by AFM (Shaiu et al. 1993a,b). Winfree et al. (1998) uses 1.4 nm nanogold-streptavidin conjugates to label DAE molecules. For these experiments, the central strand of $\mathbf{B}$ contains a $5'$ biotin group; after assembly of $\mathbf{AB}$ lattices, the solution containing DNA lattices was incubated with streptavidin-nanogold conjugates and then imaged by AFM.

Self-assembly is increasingly being recognized as a route to nanotechnology (Whitesides et al. 1991). Our results demonstrate the potential of using DNA to create self-assembling periodic nanostructures. The periodic blocks used here are composed of either two or four individual DX units. However, the number of component tiles in the repeat unit does not appear to be limited to such small numbers, suggesting that complex patterns could be assembled into periodic arrays. These patterns could be either direct targets in nanofabrication or aids to the construction of such targets. Because oligonucleotide synthesis can readily incorporate modified bases at arbitrary positions, it should be possible to control the structure within the periodic block by decoration with chemical groups, catalysts, enzymes and other proteins (Niemeyer et al. 1994), metallic nanoclusters (Alivisatos et al. 1996; Mirkin et al. 1996), conducting silver clusters (Braun et al. 1998), DNA enzymes (Breaker and Joyce 1994) or other DNA nanostructures such as polyhedra (Chen and Seeman 1991; Zhang and Seeman 1994b).

It may be possible to extend the two-dimensional lattices demonstrated here into three dimensions. Designed crystals could potentially serve as scaffolds for the crystallization of macromolecules (Seeman 1982), as photonic materials with novel properties (Joannopolous et al. 1995), as designable zeolite-like materials for use as catalysts or as molecular sieves (Ribeiro et al. 1996), and as scaffolds for the assembly of molecular electronic components (Robinson and Seeman 1987) or biochips (Haddon and Lamola 1985).

The self-assembly of aperiodic structures should also be considered. It may be possible to design molecular Wang tiles that self-assemble into aperiodic crystals according to algorithmic rules (Winfree 1996b; Winfree et al. in press). It will be crucial to understand the mechanisms of crystallogenesis and crystal growth in this system to provide a firm underpinning for theoretical proposals of computation by self-assembly.

Progress in this field will require detailed knowledge of the physical, kinetic, structural, dynamic and thermodynamic parameters that characterize DNA self-assembly. Additionally, improved methods for error reduction and purification must be developed. The approach described here provides a uniquely versatile experimental system for investigating these issues.

Bibliography

Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, 1994.

Leonard M. Adleman. On constructing a molecular computer. In Lipton and Baum (1996), pages 1–21.

A. Paul Alivisatos, Kai P. Johnsson, Xiaogang Peng, Troy E. Wilson, Colin J. Loweth, Marcel P. Bruchez, Jr., and Peter G. Schultz. Organization of 'nanocrystal molecules' using DNA. *Nature*, 382:609–611, 1996.

L. Babai, P. Pudlák, V. Rödl, and E. Szemeredi. Lower bounds to the complexity of symmetric boolean functions. *Theoretical Computer Science*, 74:313–323, 1990.

Eric Bach, Anne Condon, Elton Glaser, and Celena Tanguay. DNA models and algorithms for NP-complete problems. In *Proceedings of the 11th Conference on Computational Complexity*, pages 290–299. IEEE Computer Society Press, 1996.

David A. Barrington. Bounded width branching programs. *Ph. D. Thesis, Massachusetts Institute of Technology, TR# MIT/LCS/TR-361*, 1986.

Donald Beaver. Universality and complexity of molecular computation. *WWW preprint:* `http://www.transarc.com/afs/transarc.com/public/beaver/` `html/research/alternative/molecute/publications/psp95.ps`, 1995.

Donald Beaver. A universal molecular computer. In Lipton and Baum (1996), pages 29–36.

Charles H. Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17:525–532, 1973.

Charles H. Bennett. The thermodynamics of computation – a review. *International Journal of Theoretical Physics*, 21(12):905–940, 1982.

Robert Berger. The undecidability of the domino problem. *Memiors of the AMS*, 66:1–72, 1966.

Michael Biafore. Universal computation in few-body automata. *MIT*, preprint.

G. Binnig, C. F. Quate, and Ch. Gerber. Atomic force microscope. *Physical Review Letters*, 56 (9):930–933, 1986.

Dan Boneh, Chris Dunworth, Richard J. Lipton, and Jiří Sgall. On the computational power of DNA. *Discrete Applied Mathematics*, 71:79–94, 1996a.

Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES using a molecular computer. In Lipton and Baum (1996), pages 37–65.

R. B. Boppana and M. Sipser. The complexity of finite functions. In *Handbook of Theoretical Computer Science*. Elsevier Science Publishers B. V., 1990.

Erez Braun, Yoav Eichen, Uri Sivan, and Gdalyahu Ben-Yoseph. DNA-templated assembly and electrode attachment of a conducting silver wire. *Nature*, 391:775–778, 1998.

Ronald R. Breaker and Gerald F. Joyce. A DNA enzyme that cleaves RNA. *Chemistry & Biology*, 1:223–229, 1994.

Jin-yi Cai and Richard J. Lipton. Subquadratic simulations of circuits by branching programs. In 30th *Annual Symposium on Foundations of Computer Science*, pages 568–573. IEEE Computer Society Press, 1989.

Charles R. Cantor and Paul R. Schimmel. *Biophysical Chemistry, Part III: The behavior of biological macromolecules*. W. H. Freeman and Company, 1980.

Eugene S. Carter, II and Chang-Shung Tung. NAMOT2 — a redesigned nucleic acid modelling tool: construction of non-canonical DNA structures. *CABIOS*, 12(1):25–30, 1996.

Marvin H. Caruthers. Gene synthesis machines. *Science*, 230:281–285, 1985.

Junghuei Chen and Nadrian C. Seeman. The synthesis from DNA of a molecule with the connectivity of a cube. *Nature*, 350:631–633, 1991.

Alexander B. Chetverin and Fred Russell Kramer. Oligonucleotide arrays: New concepts and possibilities. *Bio/Technology*, 12:1093–1099, 1994.

S. N. Cohen, A. C. Y. Chang, H. W. Boyer, and R. B. Helling. Construction of biologically functional bacterial plasmids in vitro. *Proc. Nat. Acad. Sci. USA*, 70:3240–3244, 1973.

R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr. Good encodings for DNA-based solutions to combinatorial problems. In Landweber and Lipton (in press).

Peggy S. Eis and David P. Millar. Conformational distributions of a four-way junction revealed by time-resolved flourescence resonance energy transfer. *Biochemistry*, 32(50):13852–13860, 1993.

H. P. Erickson. Self-assembly and nucleation of a two-dimensional array of protein subunits. In *Electron Microscopy at Molecular Dimensions*, pages 309–317. Baumeister & Vogell, 1980.

Tsu-Ju Fu, Börries Kemper, and Nadrian C. Seeman. Cleavage of double-crossover molecules by T4 endonuclease VII. *Biochemistry*, 33:3896–3905, 1994.

Tsu-Ju Fu and Nadrian C. Seeman. DNA double-crossover molecules. *Biochemistry*, 32:3211–3220, 1993.

Peter Gacs and John Reif. A simple three-dimensional real-time reliable cellular array. *Journal of Computer and System Sciences*, 36(2):125–147, 1988.

Martin Gardner. Pascal's triangle. *Scientific American*, 215(6):128–132, 1966.

Seymour Ginsburg. *The Mathematical Theory of Context Free Languages*. McGraw-Hill, 1966.

Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman and Company, New York, 1986.

R. C. Haddon and A. A. Lamola. The molecular electronic device and the biochip computer: present status. *Proc. Nat. Acad. Sci. USA*, 82:1874–1878, 1985.

Masami Hagiya, Masanori Arita, Daisuke Kiga, Kensaku Sakamoto, and Shigeyuki Yokoyama. Towards parallel evaluation and learning of boolean μ-formulas with molecules. In Wood (in press).

William Hanf. Nonrecursive tilings of the plane I. *The Journal of Symbolic Logic*, 39:283–285, 1974.

H. G. Hansma, J. Vesenka, C. Siegerist, G. Kelderman, H. Morrett, R. L. Sinsheimer, V. Elings, C. Bustamante, and P. K. Hansma. Reproducible imaging and dissection of plasmid DNA under liquid with the atomic force microscope. *Science*, 256:1180–1184, 1992.

J. Hardy, O. de Pazzis, and Y. Pomeau. Molecular dynamics of a classial lattice gas: transport properties and time correlation functions. *Phys. Rev. A*, 13:1949–1960, 1976.

Alexander J. Hartemink and David K. Gifford. Thermodynamic simulation of deoxynucleotide hybridization for DNA computation. In Wood (in press).

John J. Hopfield. Kinetic proofreading: a new mechanism for reducing errors in biosynthetic processes requiring high specificity. *Proc. Nat. Acad. Sci. USA*, 71(10):4135–4139, 1974.

John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proc. Nat. Acad. Sci. USA*, 79(8):2554–2558, 1982.

J. D. Joannopolous, R. D. Meade, and J. N. Winn. *Photonic crystals: moulding the flow of light*. Princeton University Press, Princeton, 1995.

Nataša Jonoska, Stephen A. Karl, and Masahico Saito. Creating 3-dimensional graph structures with dna. In Wood (in press).

Z. Kam, A. Shaikevitch, H. B. Shore, and G. Feher. Crystallization processes of biological macro-molecules. In *Electron Microscopy at Molecular Dimensions*, pages 302–308. Baumeister & Vogell, 1980.

Lila Kari, editor. *Proceedings of the 4th DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 16-19, 1998*, in press .

Richard M. Karp, Claire Kenyon, and Orli Waarts. Error-resilient DNA computation. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 458–467. SIAM, 1996.

Donald E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed)*. Addison-Wesley, 1973.

Arthur Kornberg and Tania A. Baker. *DNA Replication (2nd ed.)*. W. H. Freeman and Company, 1991.

R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 3:183–191, 1961.

Laura Landweber and Richard Lipton, editors. *Proceedings of the 2nd DIMACS Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., Providence, RI, in press. American Mathematical Society.

Thomas H. Leete, Matthew D. Schwartz, Robert M. Williams, David H. Wood, Jerome S. Salem, and Harvey Rubin. Massively parallel DNA computations: Expansion of symbolic determinants. In Landweber and Lipton (in press).

Xiaojun Li, Xiaoping Yang, Jing Qi, and Nadrian C. Seeman. Antiparallel DNA double crossover molecules as components for nanoconstruction. *Journal of the American Chemical Society*, 118 (26):6131–6140, 1996.

K. Lindgren and M. Nordahl. Universal computation in simple one-dimensional cellular automata. *Complex Systems*, 4(3):299–318, 1990.

Richard J. Lipton. DNA solutions of hard computational problems. *Science*, 268:542–544, 1995.

Richard J. Lipton. Dna computations can have global memory. unpublished manuscript, 1996a.

Richard J. Lipton. Speeding up computations via molecular biology. In Lipton and Baum (1996), pages 67–74.

Richard J. Lipton and Eric B. Baum, editors. *DNA Based Computers: Proceedings of a DIMACS Workshop, April 4, 1995, Princeton University*, volume 27 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, Providence, RI, 1996. American Mathematical Society.

Bing Liu, Neocles B. Leontis, and Nadrian C. Seeman. Bulged 3-arm DNA branched junctions as components for nanoconstruction. *Nanobiology*, 3:177–188, 1994.

Rong-Ine Ma, Neville R. Kallenbach, Richard D. Sheardy, Mary L. Petrillo, and Nadrian C. Seeman. Three-arm nucleic acid junctions are flexible. *Nucleic Acids Research*, 14:9745–9753, 1986.

A. J. Malkin, Yu. G. Kuznetsov, T. A. Land, J. J. DeYoreo, and A. McPherson. Mechanisms of growth for protein and virus crystals. *Nature Structural Biology*, 2(11):956–959, 1995.

Norman Margolus. Physics-like models of computation. *Physica 10D*, pages 81–95, 1984.

Christoph Meinel. *Modified Branching Programs and Their Computational Power, LNCS 370*. Springer-Verlag, 1989.

Chad A. Mirkin, Robert L. Letsinger, Robert C. Mucic, and James J. Storhoff. A DNA-based method for rationally assembling nanoparticles into macroscopic materials. *Nature*, 382:607–609, 1996.

Dale Myers. Nonrecursive tilings of the plane II. *The Journal of Symbolic Logic*, 39:286–294, 1974.

Christof M. Niemeyer, Takeshi Sano, Cassandra L. Smith, and Charles R. Cantor. Oligonucleotide-directed self-assembly of proteins. *Nucleic Acids Research*, 22:5530–5539, 1994.

Igor V. Ouporov and Neocles B. Leontis. Refinement of the solution structure of a branched DNA three-way junction. *Biophysical Journal*, 68:266–274, 1995.

Qi Ouyang, Peter Kaplan, Shumao Liu, and Albert Libchaber. DNA solution of the maximal clique problem. *Science*, 278:446–449, 1997.

Roger Penrose. Pentaplexity. *Eureka*, 39:16–22, 1978.

Mary L. Petrillo, Colin J. Newton, Richard P. Cunningham, Rong-Ine Ma, Neville R. Kallenbach, and Nadrian C. Seeman. The ligation and flexibility of four-arm DNA junctions. *Biopolymers*, 27:1337–1352, 1988.

Pavel Pudlák. The hierarchy of boolean circuits. *Computers and Artificial Intelligence*, 6(4): 449–468, 1987.

Hangxia Qiu, John C. Dewan, and Nadrian C. Seeman. A DNA decamer with a sticky end: The crystal structure of d-CGACGATCGT. *Journal of Molecular Biology*, 267:881–898, 1997.

Robin S. Quartin and James G. Wetmur. Effect of ionic strength on the hybridization of oliodeoxynucleotides with reduced charge due to methylphosphonate linkages to unmodified oligodeoxynucleotides containing the complementary sequence. *Biochemistry*, 28:1040–1047, 1989.

Alexander A. Razborov. Lower bounds for deterministic and nondeterministic branching programs. In *Fundamentals of Computation Theory, LNCS 529*, pages 47–60. Springer-Verlag, 1991.

John Reif. Local parallel biomolecular computing. In Wood (in press).

D. Rhodes and A. Klug. Helical periodicity of DNA determined by enzyme digestion. *Nature*, 286:573–578, 1980.

F. Ramôa Ribeiro, F. Alvarez, C. Henriques, F. Lemos, J. M. Lopes, and M. F. Ribeiro. Structure-activity relationships in zeolites. *Journal of Molecular Catalysis A: Chemical*, 96:245–270, 1996.

Bruce H. Robinson and Nadrian C. Seeman. The design of a biochip: A self-assembling molecular-scale memory device. *Protein Engineering*, 1(4):295–300, 1987.

Raphael M. Robinson. Undecidability and nonperiodicity of tilings of the plane. *Inventiones Math.*, 12:177–909, 1971.

Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing Machines. In Lipton and Baum (1996), pages 75–119.

Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W. K. Rothemund, and Leonard M. Adleman. A sticker based architecture for DNA computation. In Landweber and Lipton (in press).

Kensaku Sakamoto, Daisuke Kiga, Ken Momiya, Hidetaka Gouzu, Shigeyuki Yokoyama, Shuji Ikeda, Hiroshi Sugiyama, and Masami Hagiya. State transitions with molecules. In Kari (in press).

John SantaLucia, Jr., Hatim T. Allawi, and P. Ananda Seneviratne. Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry*, 35(11):3555–3562, 1996.

Anthony Schwacha and Nancy Kleckner. Identification of double Holliday junctions as intermediates in meiotic recombination. *Cell*, 83:783–791, 1995.

Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99(2): 237–247, 1982.

Nadrian C. Seeman. *De novo* design of sequences for nucleic acid structural engineering. *Journal of Biomolecular Structure & Dynamics*, 8(3):573–581, 1990.

Wen-Ling Shaiu, Drena D. Larson, James Vesenka, and Eric Henderson. Atomic force microscopy of oriented linear DNA molecules labelled with 5nm gold spheres. *Nucleic Acids Research*, 21 (1):99–103, 1993a.

Wen-Ling Shaiu, James Vesenka, Danial Jondle, Eric Henderson, and Drena D. Larson. Visualization of circular DNA molecules labelled with colloidal gold spheres using atomic force microscopy. *Journal of Vacuum Science and Technology A*, 11(4):820–823, 1993b.

Rakesh Kumar Sinha and Jayram S. Thathachar. Efficient oblivious branching programs for threshold functions. In *Proceedings of the 35th Symposium on Foundations of Computer Science*, pages 309–317, 1994.

A. R. Smith, III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18:339–353, 1971.

Warren D. Smith. DNA computers in vitro and in vivo. In Lipton and Baum (1996), pages 121–185.

Paul J. Steinhardt and Stellan Ostlund, editors. *The Physics of Quasicrystals*. World Scientific, Singapore, 1987.

Willem P. C. Stemmer, Andreas Crameri, Kim D. Ha, Thomas M. Brennan, and Herbert L. Heyneker. Single-step assembly of a gene and entire plasmid from large numbers of oligodeoxyribonucleotides. *Gene*, 164(1):49–53, 1995.

Shaojian Sun, Rachel Brem, Hue Sun Chan, and Ken A. Dill. Designing amino acid sequences to fold with good hydrophobic cores. *Protein Engineering*, 9(1):1205–1213, 1996.

Christopher Y. Switzer, Simon E. Moroney, and Steven A. Benner. Enzymatics recognition of the base-pair between isocytidine and isoguanosine. *Biochemistry*, 32(39):10489–10496, 1993.

Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines*. MIT Press, 1987.

Boris K. Vainshtein. *Modern Crystallography, Volume 1: Fundamentals of Crystals*. Springer-Verlag, New York, 1994.

Hao Wang. Dominoes and the AEA case of the decision problem. In *Proc. Symp. Math. Theory of Automata*, pages 23–55, New York, 1963. Polytechnic Press.

J. C. Wang. Helical repeat of DNA in solution. *Proc. Nat. Acad. Sci. USA*, 76:200–203, 1979.

Ingo Wegener. *The Complexity of Boolean Functions*. John Wiley & Sons, 1987.

James G. Wetmur. DNA probes: Applications of the principles of nucleic acid hybridization. *Critical Reviews in Biochemistry and Molecular Biology*, 36:227–259, 1991.

George M. Whitesides, John P. Mathias, and Christopher T. Seto. Molecular self-assembly and nanochemistry: a chemical strategy for the synthesis of nanostructures. *Science*, 254:1312–1319, 1991.

Erik Winfree. Complexity of restricted and unrestricted models of molecular computation. In Lipton and Baum (1996), pages 187–198.

Erik Winfree. On the computational power of DNA annealing and ligation. In Lipton and Baum (1996), pages 199–221.

Erik Winfree. Simulations of computing by self-assembly. In Kari (in press).

Erik Winfree. Whiplash PCR for $O(1)$ computing. In Kari (in press).

Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, to appear, 1998.

Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Landweber and Lipton (in press).

Stephen Wolfram. Geometry of binomial coefficients. *American Mathematical Monthly*, 91:566–571, 1984.

Stephen Wolfram. Minimal cellular automaton approximations to continuum systems. In *Cellular Automata and Complexity*, pages 329–358. Addison Wesley, 1994. Originally presented at Cellular Automata '86.

David Wood, editor. *Proceedings of the 3rd DIMACS Meeting on DNA Based Computers, held at the University of Pennsylvania, June 23-25, 1997*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., Providence, RI, in press. American Mathematical Society.

Kaizhi Yue and Ken A. Dill. Inverse protein folding problem - designing polymer sequences. *Proc. Nat. Acad. Sci. USA*, 89(9):4163–4167, 1992.

J. Yunes. Seven-state solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 127:313–332, 1994.

Siwei Zhang and Nadrian C. Seeman. Symmetric holliday junction crossover isomers. *Journal of Molecular Biology*, 238:658–668, 1994a.

Yuwen Zhang and Nadrian C. Seeman. The construction of a DNA truncated octahedron. *Journal of the American Chemical Society*, 116:1661–1669, 1994b.