DNA Computation: Theory, Practice, and Prospects

C. C. Maley*
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
545 Technology Sq.
Cambridge, MA 02139
617-253-6567
cmaley@ai.mit.edu

January 26, 1998

To appear, with modifications, in Evolutionary Computation.

Abstract

Adleman launched the field of DNA computing with a demonstration in 1994 that strands of DNA could be used to solve the Hamiltonian Path Problem for a simple graph (Adleman, 1994). He also identified three broad categories of open questions for the field. First, is DNA capable of universal computation? Second, what kinds of algorithms can DNA implement? Third, can the error rates in the manipulations of the DNA be controlled enough to allow for useful computation? In the two years that have followed, theoretical work has shown that DNA is in fact capable of universal computation (Beaver, 1995; Rothemund, 1996; Winfree, 1995; Boneh et al., 1995b; Csuhaj-Varjú et al., 1996). Furthermore, algorithms for solving interesting questions, like breaking the Data Encryption Standard (Boneh et al., 1995a), have been described using currently available technology and methods. Finally, a few algorithms have been proposed to handle some of the apparently crippling error rates in a few of the common processes used to manipulate DNA (Karp et al., 1995; Boneh and Lipton, 1996; Amos et al., 1996). I am inclined to agree with Lipton (Lipton, 1995) that it is thus unlikely that DNA computation is doomed to be "only a curious footnote to the history of computing." However, much work remains to be done on the containment and correction of errors. It is far from clear if the problems in the error rates can be solved sufficiently to ever allow for general purpose computation that will challenge the more popular substrates for computation.

Unfortunately, biological demonstrations of the theoretical results have been sadly lacking. To date, only the simplest of computations have been carried out in DNA

*I am grateful to Andrew Blumberg for his perspective on these topics and pointers into the literature. I would also like to thank Marvin Minsky, Lynn Stein, and Gill Pratt for giving me the opportunity to explore DNA computing.

(Adleman, 1994; Guarnieri et al., 1996; Cai et al., 1996; Liu et al., 1996; Winfree et al., 1996). To make significant progress, the field will require both the assessment of the practicality of the different manipulations of DNA (Kaplan et al., 1995) and the implementation of algorithms for realistic problems. Theoreticians, in collaboration with experimentalists, can contribute to this research program by settling on a small set of practical and efficient models for DNA computation.

Key words: DNA Computation, Molecular Computation.

1 In the Beginning

In November of 1994, Adleman published a dramatic reminder that computation is independent of any particular substrate (Adleman, 1994). By using strands[1] of DNA annealing to each other, he was able to compute a solution to an instance of the Hamiltonian Path Problem (HPP). While the concept of using DNA to do computation had been presaged by work in formal language theory (Head, 1987) and artificial selection of RNA (Sassanfar and Szostak, 1993; Lorsch and Szostak, 1994; Beaudry and Joyce, 1992; Stemmer, 1994), these precursors had largely gone unnoticed in main stream computer science. Adleman's work sparked off intense excitement and marked the birth of a new field, DNA computation.

How useful will DNA computing[2] be in the future? This paper will review the current practice and theory of the field so as to give the reader a basis upon which to answer that question. The optimistic hope is that DNA will replace silicon as a medium for universal computing. The pessimistic view is that the engineering problems will make the medium impractical for useful computation. It may be the case that elegant algorithms on silicon computers will always perform better than algorithms on a DNA computer. I take the position that DNA computation is likely to turn out to have a role similar to neural networks. They will be better than traditional methods for some problems, and worse for others. In this way, DNA computation would complement rather than replace our current computational techniques.

1.1 ...there was Adleman

To Adleman's eye, there is computation in the interaction of molecules. His insight was that the sequence specific binding properties of DNA allows it to carry out massively parallel computation in a test tube. The Hamiltonian Path Problem is a famous NP-complete problem in computer science. The question is, given a graph, a starting node, and an ending node, is there a path through the graph beginning and ending at the specified nodes such that every node in the graph is visited exactly once? Briefly, Adleman's experiment followed these six steps:

0. Create a unique sequence of 20 nucleotides (a 20-mer) associated with each vertex in the graph. Similarly, create 20-mer sequences to represent the edges in a graph. If two vertex sequences, v_1 and v_2, were each composed of two 10- mer sequences x_1y_1 an x_2y_2 then an edge, $e_{1,2}$, from v_1 to v_2 was a 20-mer composed of the Watson-Crick complement[3] of y_1 and x_2 ($\overline{y_1x_2}$). Thus, when put in solution the edge sequence will

[1] I have attempted to either avoid excessive jargon, like oligomers for "strands", or explain it as I use it. Unfortunately, Biology is notorious for its oversized lexicon.

[2] DNA computing is by rights a subfield of molecular computing. There is no particular reason to constrain research to manipulating DNA. However, since little work in molecular computing has been done other molecules, this review will focus narrowly upon DNA computing.

[3] The four nucleic acids that make up DNA combine in "Watson-Crick" complementary pairs. Thus Adenine (A) binds to Thymine (T) while Guanine (G) binds to Cytosine (C). Any sequence of nucleotides, such as TAGCC, has a complementary sequence, ATCGG, that will bind to it so as to form a double strand of DNA.

bind to the second half of the source vertex and the first half of the destination vertex, and thus serve as a sort of "connector" for binding the vertices together.

1. Generate random paths through the graph by putting many copies of the vertex and edge sequences into a solution and letting them anneal to each other. That is, any complementary sequences will bind to each other to form double stranded DNA. In addition to the simple edge and vertex sequences, two further sequences were added. If $v_{\text{in}} = x_{\text{in}}y_{\text{in}}$ was the starting vertex and $v_{\text{out}} = x_{\text{out}}y_{\text{out}}$ was the terminating vertex, the sequences $\overline{x_{\text{in}}}$ and $\overline{y_{\text{out}}}$ were added in order to "cap off" sequences starting with v_{in} and ending with v_{out}. The process of annealing resulted in long double strands of DNA with breaks in the "backbone" wherever one vertex (or edge) sequence ended and another began. These breaks were filled in by adding ligase enzyme to the solution, thus forming the covalent bonds between subsequences necessary to construct two coherent single strands of DNA annealed together.

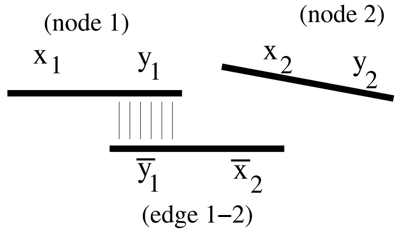

Figure 1: Paths through a graph are formed spontaneously through the annealing of segments of DNA representing nodes and edges. This shows how an edge from node 1 to 2 can "glue" together the sequences representing nodes 1 and 2. Once they are adjacent, a ligase enzyme will bind the two node sequences together. The result is a double strand of DNA, one strand representing a sequence of nodes and the complementary strand representing a sequence of edges.

2. Separate out all paths starting with v_{in} and ending with v_{out}. This was done not by actually separating the DNA representing good paths from DNA representing bad paths, but by exponentially replicating the good paths using a polymerase chain reaction (PCR). In this way, the proportion of good paths in the solution far out weighed the proportion of bad paths.

3. Separate out all paths that go through exactly n vertices, where $n = 7$ is the number of vertices in Adleman's graph. This was implemented by separating the strands by their lengths and identifying all 140-mers. Separation by length is typically carried out by gel electrophoresis. Different length strands of DNA move at different rates through an electrical field in a gel.[4] The section of the gel corresponding to 140-mers was identified, the DNA extracted from the gel, and replicated, or amplified, by PCR. This sequence of gel electrophoresis, extraction and amplification was repeated several times in order to purify the sample.

[4]For a strand of length $N < 150$, biologists have experimentally found that the distance a DNA strand travels is $d = a - b\,ln(N)$, for some constants a and b.

4. Separate out all paths that go through all n vertices. This was done by first constructing, for each vertex in the graph, a set of magnetic beads attached to the complement sequence of the vertex. The double stranded DNA from step 3 was "melted" to reduce it to a solution of single stranded DNA, and then mixed with the magnetized vertex complement sequence for vertex 1. The sequences binding to these beads were magnetically extracted. They should represent paths that include vertex 1. The process was repeated on the results of the previous extraction for each vertex in the graph. At the end, only those paths that include all 7 vertices should remain.

5. Detect if there are any DNA sequences in the remaining solution. This was done by first amplifying the results with PCR and then using gel electrophoresis to see if anything of the appropriate length is left.

Elegant in design, clumsy in implementation. This is perhaps appropriate for the first baby steps of a field. The time required to create a desired sequence depends on the enzymes used. However, it is on the scale of hours (Gannon and Powell, 1991). There are $O(n^2)$ such sequence generation processes in step 0. In many cases, subsequences might be taken from a pre-existing DNA library or bought from a vendor. Adleman omits a description of how he obtained the sequences representing the vertices, except to say that the sequences were random. The process of annealing (step1) requires about 30 seconds (Vahey et al., 1995, p. 20). Each PCR process (step 3, multiple times, and step 5) takes approximately 2 hours (Vahey et al., 1995, p. 20). The time required for gel electrophoresis (steps 2 and 5) depends on the gel used and the size and charge of the molecules. For the agarose gel that Adleman used, electrophoresis takes about 5 hours. For a polyacrylamide gel, it takes about 70 minutes (Kaplan et al., 1995). It should also be noted that gel electrophoresis requires human intervention to visually identify and extract the desired results. Constructing the magnetic beads bound to premade sequences is a simple annealing operation. Finally, separation using the magnetic beads, which Adleman noted was the most labor intensive part of the algorithm, performed n times in step 4, requires at a minimum approximately 1 hour using standard techniques (Lönneborg et al., 1995, p. 445). The entire experiment took Adleman 7 days of lab work. Adleman asserts that the time required for an entire computation should grow linearly with the size of the graph. This is true as long as the creation of each edge does not require a separate process.

Kaplan et al. (Kaplan et al., 1995) rightly criticize Adleman for not using a positive or a negative control. This is a serious issue in part because the only known attempt to replicate Adleman's experiment produced ambiguous results (Kaplan et al., 1995).

1.2 Go Forth and Multiply

Despite all the practical delays of the biological manipulations, the real excitement over Adleman's work comes from the fact that in step 1, a massively parallel computation of all the possible paths through the graph is carried out in O(1) time. With an estimated upper bound of 10^{20} (Adleman, 1994) "operations" in step 1, the massive parallelism of DNA computing might be able to make the delays in the other steps seem negligible. In fact, Adleman makes a series of claims for the promise of DNA computing (Adleman, 1994). DNA computation may outstrip modern silicon computers on three counts:

1. Speed. This should fall out from the massive parallelism of the approach, amortized across the slow serial operations.

2. Energy efficiency. Since the molecules actually release energy when they anneal together, there is some hope that computations could be carried out using very little energy.

3. Information density. Packing 10^{20} strands of data into a liter of volume would give us an information density at least five orders of magnitude better than current hard drive technology.

But more immediately, he identifies a number of important open questions for the new field:

1. What is the theoretical power of DNA computation?

2. Are there practical algorithms for solving problems with DNA?

3. Can errors be adequately controlled?

The purpose of this paper is to evaluate the promise of DNA computing and review the progress on these outstanding questions. Along the way I will attempt to sketch a missing implementation of one of the abstract models of DNA computation (Csuhaj-Varjú et al., 1996), add some thoughts on the implementation of genetic algorithms with DNA, and identify some of the most important remaining open questions.

2 Theory

First, a cautionary tale. The "little algorithm that can," on a graph of 7 vertices, runs into serious problems on a graph of 100 vertices. A test-tube full of 10^{20} strands of DNA performing a computation does not give us non-determinism in polynomial time (Hartmanis, 1995; Linial and Linial, 1995). Linial and Linial estimate that if the same method were to be used on a graph of 70 vertices, the experiment would require 10^{25} kilograms of nucleotides. Adleman's demonstration is not exciting because of the fact that HPP is a hard problem, but rather because he showed how to exploit the immense parallelism of molecular interactions.

2.1 Models of DNA Computation

The number of different models for DNA computing seems to be linear in the number of manuscripts. Everyone has their own way of using DNA for computation (Beaver, 1995; Rothemund, 1996; Winfree, 1995; Boneh et al., 1995b; Csuhaj-Varjú et al., 1996; Roweis et al., 1996; Cai et al., 1996, for example). This is a symptom of the fact that the field is still exploring the possibilities and has hardly begun to settle on a preferred technique for making DNA do useful computations. The models are usually built on a set of operations that are hypothesized to be realizable processes for manipulating DNA. These operations include:

Anneal This is probably the most basic operation used in DNA computing. Single stranded complementary DNA will spontaneously form a double strand of DNA when suspended in solution. This occurs through the hydrogen bonds that arise when complementary base pairs are brought into proximity. This is also called “hybridization.”

Melt The inverse of annealing is “melting.” That is, the separation of double stranded DNA into single stranded DNA. As the name implies, this can be done by raising the temperature beyond the point where the longest double strands of DNA are stable. Since the hydrogen bonds between the strands are significantly weaker than the covalent bonds between adjacent nucleotides, heating separates the two strands without breaking apart any of the sequences of nucleotides. “Melting” is a bit of a misnomer because the same effect can be achieved by washing the double stranded DNA in doubly distilled water. The low salt content also destabilizes the hydrogen bonds between the strands of DNA and thereby separates the two strands. Heating can be selectively used to melt apart short double stranded sequences while leaving longer double stranded sequences intact.

Ligate Often invoked after an annealing operation, ligation concatenates strands of DNA. While it is possible to use some ligase enzymes to concatenate free floating double stranded DNA, it is dramatically more efficient to allow single strands to anneal together, connecting up a series of single strand fragments, and then use ligase to seal the covalent bonds between adjacent fragments, as in Figure 2.

Polymerase Extension Polymerase enzymes attach to the 3’ end of a short strand that is annealed to a longer strand. It then extends the 3’ side of the shorter strand so as to build the complementary sequence to the longer strand. This is shown in Figure 3.

Cut Restriction enzymes will cut a strand of DNA at a specific subsequence. As of 1993, there were over 2300 different known restriction enzymes that were specific to more than 200 different subsequences. These subsequences are usually on the order of 4 to 8 nucleotides. Some restriction enzymes will only cleave single stranded DNA, while others will only cleave double stranded DNA. Similarly, methylation of the Cytosine nucleotides on a strand of DNA interferes with the activity of some restriction enzymes, but not others. Furthermore, some restriction enzymes will cleave a wide range of subsequences of DNA. That is, the specificity of activity varies across different enzymes.

Destroy Subsets of strands can be systematically destroyed, or “digested” by enzymes that preferentially break apart nucleotides in either single or double stranded DNA.

Merge Given two test tubes, combine them, usually by pouring one into the other.

Separate by Length Given a test tube of DNA, split it into two test tubes, one with strands of a particular length, and the other with all the rest. This is done with gel electrophoresis and requires the DNA to be extracted from the gel once the strands of different length have been identified by some form of staining or radioactive tagging.

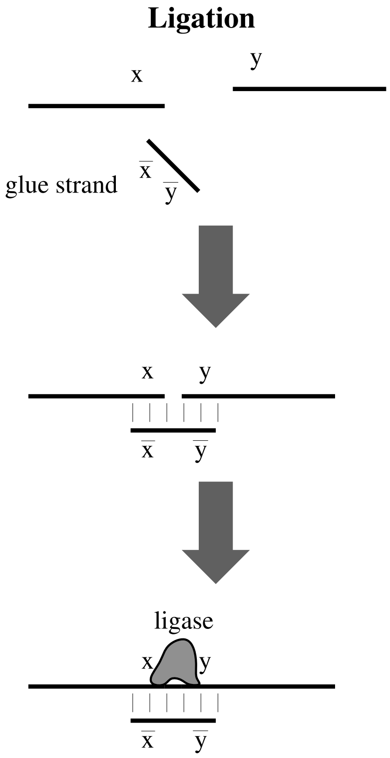

Figure 2: In order to connect strand y to strand x a "glue strand" is used to bring the two strands into proximity through annealing. Then a ligase enzyme is added to fuse the two strands together.

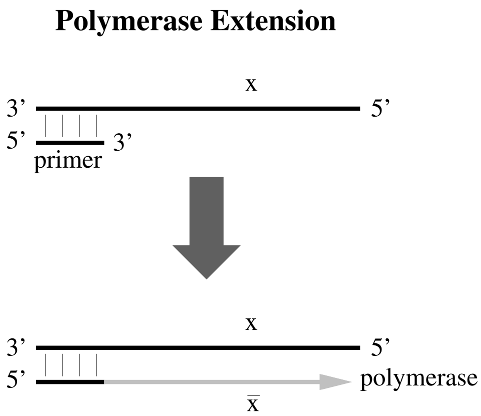

Figure 3: A polymerase enzyme attaches to the 3' end of a short primer sequence and constructs the complement ($\overline{x}$) of the longer sequence.

These basic manipulations can be combined into higher level manipulations. Perhaps the most famous example of a higher level manipulation is Polymerase Chain Reaction (PCR). Composite manipulations include:

Amplify Given a test tube of DNA, make multiple copies of a subset of the strands present. This is done with polymerase chain reaction (PCR). PCR requires a beginning and an ending subsequence, called "primers" which are usually about 20 base pairs long, to identify the sequence (called the "template") to be replicated. Copies of these subsequences anneal to the single strands and polymerase enzymes build the complementary strands, as in Figure 4. Heat then melts apart the double strands, reducing them to single strands and the process repeats, doubling the number of strands in the test tube each cycle.

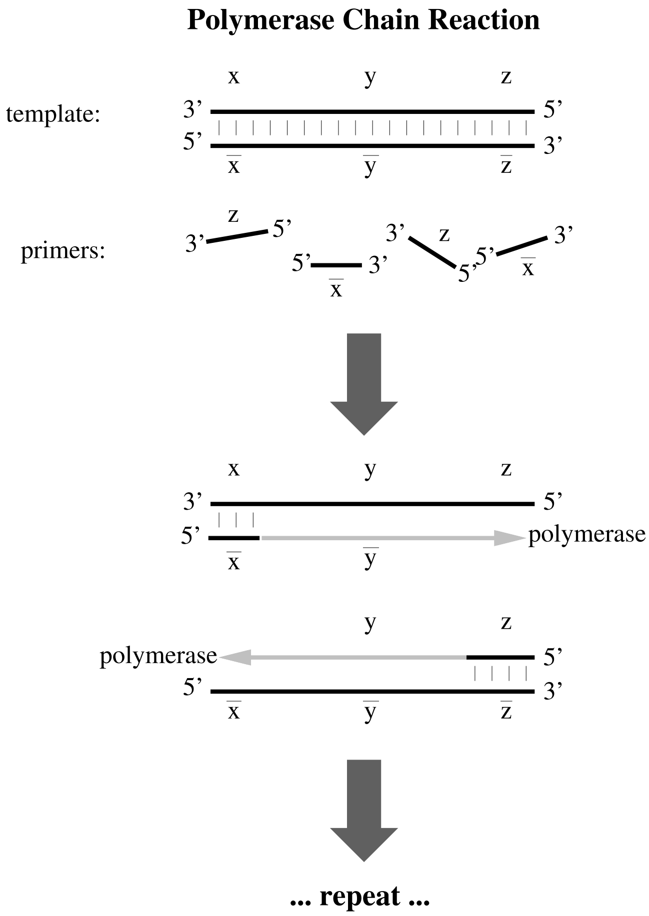

Figure 4: Polymerase Chain Reaction (PCR) follows in cycles of 3 steps. 1. The double stranded templates are melted apart. 2. The primers anneal to both strands. 3. A polymerase enzyme extends the primers into replicas of the templates. This is repeated, causing an exponential growth in the number of templates, as long as there are enough primers in the solution to catalyze the reaction. Note that because polymerase attaches to the 3' end of a primer we need to use the subsequences $\overline{x}$ and z to get the desired reaction.

Separate by Subsequence Given a test tube of DNA, split it into two test tubes, one with the sequences that contain the specified subsequence, and the other with the rest. We have already seen how this can be done with magnetic beads and the Watson-Crick complement of the specified subsequence, as in Adleman (Adleman, 1994). This is also sometimes called an "extraction." It can also be implemented by destroying the unwanted strands.

Append This adds a specific subsequence to the ends of all the strands. This can be done by annealing a short strand to a longer strand so that the short strand extends off the end of the longer strand. Then the complement of the subsequence that is extending off the end can be added to the longer strand either through the use of a polymerase enzyme, or through the introduction and annealing of that sequence, cemented by ligase, as depicted in Figure 2.

Mark This operation tags strands so that they can be separated or otherwise operated upon selectively. Marking is commonly implemented by making a single strand into a double strand through annealing or the action of a polymerase. However, it can also mean appending a tag sequence to the end of a strand or even methylation of the DNA or (de)phosphorylation of the 5' ends of the strands.

Unmark The complement of the marking operation. Unmark removes the marks on the strands.

Most models only use a few of the possible operations. I will briefly describe three qualitatively different models in order to sample some of the flavors on the menu.

2.1.1 Generate and Test

The most popular model for using DNA to do computation comes directly out of Adleman's first paper (Adleman, 1994; Adleman, 1996; Boneh et al., 1995b; Roweis et al., 1996). The technique was refined into a process with two phases (Adleman, 1996; Boneh et al., 1995b). First, randomly generate all possible solutions to the problem at hand. Second, isolate the correct solution through repeated separations of the DNA into incorrect solutions and potentially good solutions. At the end of this series of separation steps it may only be necessary to detect if there are any DNA strands left in set of good solutions. Adleman's step 5 provides one example.

Generating solutions and then isolating the correct one was easily generalized to finding all the inputs to a boolean circuit that would result in a specified output (Boneh et al., 1995b). Lipton (Lipton, 1995) showed how to generate the set of all[5] binary strings of length n in O(1), as long as $n < 64$. Lipton's procedure involves first creating a graph of size O(n) such that a path through the graph is a realization of a binary number of length n. See Figure 5. The graph is implemented in the same way as Adleman's (Adleman, 1994). In Lipton's words,

> The graph G_n has nodes $a_1, x_1, \overline{x_1}, a_2, \ldots, a_{n+1}$ with edges from a_k to both x_k and $\overline{x_k}$ and from both x_k and $\overline{x_k}$ to a_{k+1}. ... The graph is constructed so that all

[5] I will defer the problem of guaranteeing that all strings are actually generated until Section 3.2.

paths that start at a_1 and end at a_{n+1} encode an n-bit binary number. At each stage a path has exactly two choices: If it takes the vertex with an unprimed label, it will encode a 1; if it takes the vertex with a primed label, it will encode a 0. Therefore, the path $[a_1\overline{x_1}a_2x_2a_3]$ encodes the binary number 01. [I have altered the notation to preserve internal consistency.]

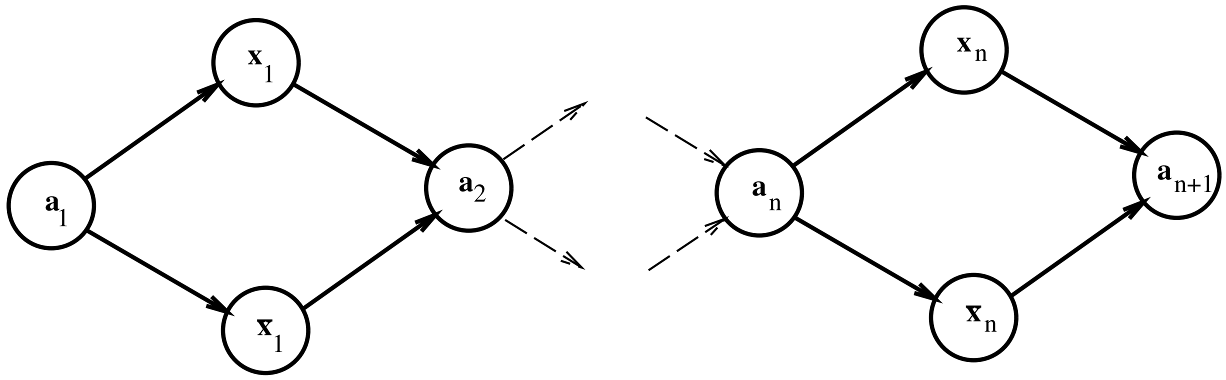

Figure 5: Lipton's graph for constructing binary numbers. The vertices and edges are constructed using Adleman's algorithm so that the longest path through the graph will represent an n bit binary number. If the path goes through an $\overline{x}$ then it has a 0 at that position, otherwise it has a 1 at that position.

A more recent implementation of bit strings allows for the modification of bits. The "sticker model" (Roweis et al., 1996) represents a 0 as a sequence of single stranded DNA and a 1 as a sequence of double stranded DNA. The basic structure is a single fixed length strand representing a string of 0's. Each M-mer interval (where M is about 20 or 30) is unique and encodes a bit position. Thus, to set a bit in all the strands in a test-tube, one merely has to add a large quantity of the subsequence representing the complement of that bit position to the test tube. A collection of random bit strings can be created by repeatedly splitting the test tube into two test tubes, setting the next bit position in one of the two tubes, and then merging the results. The problem of clearing a single bit position without clearing all the bit positions has not yet been solved and remains a limitation of the sticker model at present.

The second phase of solving a boolean circuit involves repeated separations, with a little molecular bookkeeping to keep track of the progress. The idea is to create an ordering on the gates of the circuit so that the inputs of gate g_i are either inputs to the circuit or outputs of gates g_j where $j < i$. Then using this ordering, we evaluate the output for the gates, one at a time, given the input values represented in each strand. We append to each strand a bit indicating the value of the output of each gate.[6] This is done in parallel across all the strands, representing all the possible input values. For example, if we are processing gate g_i, then we separate all strands whose inputs to gate g_i would result in the output 1. To these strands we append a subsequence representing 1 with a tag identifying it as the

[6]In the case of the sticker model, there is no "append" operation, so the base strand must be long enough to include bits for the gate evaluations.

output of gate g_i. To all the other strands we append a 0, and then we merge the two sets in order to process the next gate. Selection is done by using magnetic beads attached to the Watson-Crick complement of the subsequences representing the input bit values.[7] This process is simply repeated until the output gate has been processed. Then all strings with a 1 on the end represent input values that make the circuit output a 1.

2.1.2 SIMD Computation

Blumberg argues that rather than inventing an entirely new field of algorithms for DNA computation, we should instead design manipulations of DNA that can simulate traditional models of parallel computation (Blumberg, 1996a). This should liberate us from the restrictions of "generate and test" algorithms and open up the entire field of parallel algorithms to implementations in DNA.

Blumberg outlines the construction of a Connection Machine based on one-bit, single instruction stream, multiple data stream (SIMD) computations (Blumberg, 1996a). Under this construction, each strands of DNA is assumed to be the local memory of a single "processor." Assuming the need for redundancy to provide fault tolerance in the DNA manipulations, Blumberg specifies the use of sets of identical strands to represent a processor's memory. If we restrict a computation to just a few (k) bits, including flag bits, then all the possible different combinations of the states of those bits can be used to separate the strands into 2^k tubes. The different results of the computations are used to alter the strands in the different tubes, independently, and the results are merged back into a single test tube (Blumberg, 1996b; Blumberg, 1996a).

The algorithm for writing a bit into a specific location in a strand is slightly awkward. We iterate over all the bits in the strand, separating the strands on the state of the current bit and appending a copy to the end of the strand. When we reach the bit position of the location we want to alter, instead of copying the current state, we append the new state to the end of the strand. This is continued until an entirely new copy of the "memory" has been appended to the state of the old memory. We can then split the strands in half and remove the strands representing the old state. This takes $O(\text{size of memory})$ time but has the nice property that we can recover the state of the computation before the last write operation. To be fair, Blumberg only presents this as a possible solution for doing parallel computation, not the best solution (Blumberg, 1996a).

An interesting innovation in this work is an algorithm for doing communication between "processors" (strands) (Blumberg, 1996a), as depicted in Figure 6. The sending processors build up a message at the end of their strands that include the complement of the identification tags for both the sending and the receiving processors. The complements of these tags is followed by the actual data of the message. The messages are then cleaved from the processors. Next, processors append to the end of their strand the tag of the processor they are "willing" to accept a message from and their own identification tag. This allows the complements in the messages to anneal to these tags. The restriction to accepting messages from only one processor ensures that all of the redundant copies of a processor will receive

[7] Again, the sticker model requires a slightly different procedure. Separations are done on one bit position at a time. All those strands with a 0 in that position can anneal to the strands on the magnetic beads, and thus be separated from strands with a 1 in the same position.

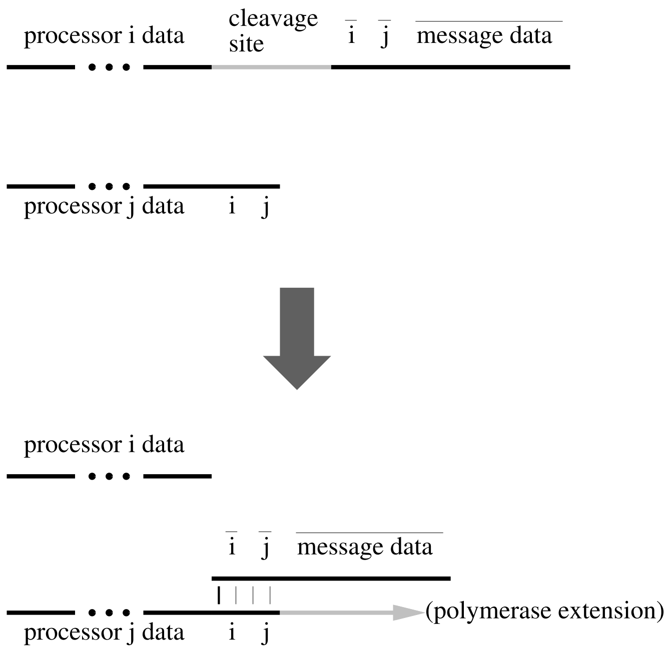

Figure 6: Processor i first appends a cleavage sequence, the complement of its own identification tag ($\bar{i}$), the complement of the identification tag for the receiving processor ($\bar{j}$), and the complement of the message ($\overline{\text{message data}}$). Processor j, the receiving processor appends the identification tags for i and j to the end of its strand. The message is then cleaved off of processor i's strand and allowed to anneal to processor j's strand. Polymerase then copies the data of the message onto the end of processor j's strand.

the same message. The addition of polymerase to the test tube should then append the complement of the message to the end of the receiving processor's strand.

2.1.3 mildly enhanced H-systems

H-systems represent a nice contrast to the more practical models introduced above. While it is hard to conceive of using an H-system implemented in DNA to do serious computation, their conceptual elegance and close relationship to computational grammars makes them ideal for research in the theory of computation with DNA.

H-systems are based on the operations of cutting and ligating strands of DNA (Csuhaj-Varjú et al., 1996). The cutting and ligating is combined into a single "splicing" operation akin to the "crossover" operation from genetic algorithms (Goldberg, 1989). The idea is that a splicing location in a string is identified by a # symbol in a string. So given two strings x#y and w#z, they may be spliced to create the strings x#z and w#y. However, these splicing operations can be restricted by the context surrounding the splicing location. More formally from (Csuhaj-Varjú et al., 1996):

Definition 1 *An extended H system is a quadruple*

$$\gamma \;=\; (V, T, A, R)$$

where V is an alphabet, $T \subseteq V$, $A \subseteq V^*$, and $R \subseteq V^*\#V^*\$V^*\#V^*$; $\#$ and $\$$ are special symbols not in V. (V is the alphabet *of γ, T is the* terminal *alphabet, A is the set of* axioms, *and R is the set of* splicing rules*; the symbols in T are called* terminals *and those in $V - T$ are called* nonterminals*.)*

For $x, y, z, w \in V^*$ and $r = u_1\#u_2\$u_3\#u_4$ in R we define

$$\begin{aligned}(x, y) \vdash_r (z, w) \quad \textit{if and only if} \quad & x = x_1u_1u_2x_2, y = y_1u_3u_4y_2, \textit{ and} \\ & z = x_1u_1u_4y_2, w = y_1u_3u_2x_2, \\ & \textit{for some } x_1, x_2, y_1, y_2 \in V^*.\end{aligned}$$

The strings x, y are called the terms *of the splicing; u_1u_2 and u_3u_4 are called the* sites *of the splicing.*

The language generated by an extended H system is the set of all the strings it can produce by repeated splicing operations such that the results consist entirely of terminal symbols. That is, the intersection of the strings it can produce with the set T^*. The H-system is called "extended" due to this final restriction that the language be a subset of T^*. We will show in Section 2.1.4 that with a mild enhancement, the addition of some context restrictions on the rules in R, the H-system is capable of universal computation.

Csuhaj-Varjú et al. do not attempt to describe an implementation for an extended H-system. The main difficulty comes in developing a non-reversible splicing operation that can be repeated at the same location in a string. If we could add the condition that only one splicing event can take place at a site in the string, then we could use two endonuclease enzymes that recognize slightly different restriction sites but leave the same "sticky" ends. For example *Bam*Hl recognizes the site GGATCC and cleaves it into G and GATCC with the middle four (GATC) left as a single "sticky" strand. *Bgl*ll recognizes AGATCT and cleaves it into A and GATCT where again the middle four (GATC) are left as a sticky end. Thus, when the product of a *Bam*Hl and a *Bgl*ll restriction anneal (due to the complementary sticky ends on the opposite strands of DNA) and are ligated, the resulting sequence GGATTT could not be cut by either enzyme, and so the splicing could not be reversed. On the other hand, a new splicing using those two enzymes also couldn't be repeated at that site.

To include the dynamic of allowing multiple splicings at one site, but only when they are desired, we will have to add a manual, though easily automated component. There will be one restriction sequence, called r. The endonuclease used will only cleave double stranded DNA with that sequence, not single stranded DNA. Such enzymes exist (Brown, 1995). Let every character in a string be separated by the sequence r. The sequence r corresponds to a # symbol in the H-system rules. For every rule $u_1\#u_2\$u_3\#u_4$ in R we introduce short complementary sequence $\overline{u_1\#u_2}$ and $\overline{u_3\#u_4}$ to the solution. By cycling the heat we allow repeated annealings, cuttings, and melting away into single strands. The problem is that with the newly generated $\overline{u_1\#u_4}$ and $\overline{u_3\#u_2}$ hanging around, they can anneal and facilitate a splicing that will undo the progress of the system. We introduce a column to the test tube, with sequences attached to it that will vacuum up this annoying trash. Specifically, the column has the sequences $u_1\#u_4$ and $u_3\#u_2$ attached to it. The column is periodically removed from the solution, rinsed with distilled water to remove the trash, and replaced in the computing solution. We would have to keep the temperature of the computing solution above the point where $\overline{u_1\#u_2}$ might anneal to $u_1\#u_4$, due to the complementary u_1. Furthermore, to prevent the restriction enzymes from cleaving the double strands attached to the column, the r sequences attached to the column should be methylated. Some restriction enzymes are inhibited by methylation at the restriction site (Nelson et al., 1993).

In order to select out the strings that are composed entirely of terminal symbols we might use either a "separation by subsequence" operation, pulling out those strings that have non-terminals, or we might add a restriction enzyme that would attack subsequences representing non-terminals. After such degradation we could either separate by length or by extracting the subsequences left with "sticky ends" from the actions of the enzymes. In this way, we should attain a "one-pot" easily automated universal computing system.

Of course, there are probably many practical challenges involved in the use of restriction enzymes[8] that would probably prevent the direct implementation of the preceding proposal. However, it is intended as an optimistic sketch of a solution. There were over 2300 different restriction enzymes known in 1993 (Roberts and Macelis, 1993), with over 200 different specificities (recognition sequences). Furthermore, there is a great diversity of details in their behaviors. Some are inhibited by methylation at the recognition sites, others will cleave DNA that contains a sequence one or more base pairs different from their specificity (Roberts and Macelis, 1993; Nelson et al., 1993). Thus, there is a fair amount of flexibility in their use for the development of a real splicing system.

[8]For example, high temperatures deactivate most restriction enzymes except for those that have evolved in thermophillic archaebacteria.

2.1.4 Is DNA Universal?

Lipton showed that Adleman's technique for finding a solution to HPP could be generalized to solve the problem of finding a satisfying assignment of an arbitrary directed contact network (Lipton, 1995). This problem includes the important specific NP-complete problem of solving an arbitrary boolean formula (SAT). But, the big question on everyone's minds was whether or not DNA computation is universal. That is, is it possible to construct a DNA computer that can compute any given (computable) function?

In short, yes.

The question has been answered multiple times by constructing an implementation of a known universal system in some model of DNA computation. Just pick your favorite universal computing abstraction: Turing Machines (Beaver, 1995; Rothemund, 1996), Blocked Cellular Automata (Winfree, 1995), boolean circuits (Boneh et al., 1995b), and type-0 Chomsky Grammars (Csuhaj-Varjú et al., 1996). Both the Blocked Cellular Automata model (which only requires the annealing and ligation operations) and the type-0 Chomsky Grammar abstractions have the added attraction of "one-pot" computation. That is, you just throw all the ingredients into a test tube, wait a little while, and then (somehow) read out the output. There are no repeated cycles of PCR, gel electrophoresis, or other labor intensive biological manipulations.

2.2 Examples of Algorithms

DNA may be used to simulate a universal type-0 Chomsky grammar, or a Turing Machine, but no one wants to program a Turing Machine. Are there practical algorithms for solving problems with DNA? Again, the answer is yes, although it remains to be seen how common such algorithms are.

2.2.1 Content-Addressable Memory

Baum (Baum, 1995) recognized that the "separation by subsequence" operation can be used to build a content-addressable memory on an unseen scale. If an address string is attached to a "content" string, a more traditional memory structure is possible, although slow. Using the "traditional" estimate of 10^{20} DNA strands in a container, the storage capabilities of such a system would vastly overwhelm current data storage media. Baum notes that copying and merging such data bases would be relatively trivial. On the other hand, Baum does not address the problem of errors in "separation by subsequence" which would form the basis of his "read" operation. Nor does he discuss the phenomena of degradation of the sequences. Strands of DNA tend to break over time. Still, it remains an intriguing and seemingly plausible suggestion.

2.2.2 Genetic Algorithms

A significant amount of work in "in vitro evolution" predated Adleman's experiment (Sassanfar and Szostak, 1993; Lorsch and Szostak, 1994; Beaudry and Joyce, 1992, for example.). This process typically involves the repeated selection of RNA molecules for some desired catalytic activity. Like most of our genetic algorithms (GA's), RNA simplifies the task by combining genotype and phenotype in one molecule. The challenge is to isolate the "more" active molecules from the "less" active ones, each generation. In vitro evolution works particularly well if you want to evolve RNA to bind to some molecule. Then selection can proceed by dipping a column covered with the target molecule into solution and pulling out all those that bind to it. You can even evolve greater specificity by adding decoy molecules, similar to the target, into the solution. Then, the RNA that is not sufficiently specific in its binding properties will bind to the decoys instead of the column, and so will not be selected from the solution. In this way, the fitness function and selection are combined in one operation. A phase of error prone amplification produces the next generation from the selected RNA strands. A very few generations (e.g., 8) with large populations (e.g., 10^{14}) are all that are necessary to evolve the desired results (Sassanfar and Szostak, 1993).

Blumberg has sketched an implementation of a more traditional GA using standard DNA computational techniques (Blumberg, 1996b). His SIMD approach, from Section 2.1.2, can be used to apply a fitness function in parallel across all the strands in the population. You can get mutation "for free" from error-prone PCR. The remaining problems are the implementation of selection for the strands with high fitness scores and the implementation of crossover.

By iterating over the bits of the fitness score, it is possible to pull out all strands that have a score above some fitness cutoff (Blumberg, 1996b). Errors in the separation manipulations give some variation in the results of a selection operation. Blumberg also suggests a method of achieving a uniform crossover.

1. Make a copy of the parental DNA for use in crossover.

2. Strands of DNA are first paired up using ligation. Although Blumberg is sparse on the details of how to do this, it should be possible to split the population randomly into two test tubes by simply dividing the volume in half. Then in one half you can append a "sticky" strand to the ends of the strands, and in the other half you can append the complementary "sticky" strand to the beginning of the strands. When the two tubes are merged they will spontaneously form random pairs.

3. Uniform crossover proceeds by iterating over the bits of both parents and building up a new strand of DNA appended to the end of the pair. For each bit position the parental pairs are separated into three test tubes, one for the parents that both have a 1 in that position, one for the parents that both have a 0, and one for parents that have different bits at the current position. In the last case, a random bit is appended to the new "offspring" strand. In the other two cases either a 1 or a 0 is appended as appropriate (Blumberg, 1996b).

4. Cut the newly constructed strands from the pairs of parents and merge them with the original tube of parental DNA.

Unfortunately, uniform crossover with a 50% recombination probability between every bit tends to destroy any partial solutions, or "building blocks," that the GA might have developed. An alternative, one-point crossover procedure might be realized through the following protocol. Recall that the encoding of the bit strings include subsequences representing the bits as well as subsequences representing the index or bit position of the following bit.

1. Split the "parental" double-stranded DNA into two test tubes.

2. Add primers to both test tubes for only one of the two strands in each double-stranded pair of DNA.

3. Heat and allow polymerase to construct a partial copy of those strands.

4. Pull out the strands containing the primers. These should be partial fragments of the original strands.

5. Switch these partial fragments between test tubes. Since the subsequences representing the bit positions will anneal with the same subsequences in the new parental strands, many of the partial fragments can still act as primers for the polymerase.

6. Allow polymerase to complete the construction of the strands on the new templates.

This will result in a bias for "mating" between strands that are similar in the region of the point of crossover due to the preferential annealing of a partial strand to a well matching parent in the new test tube during step 5.

In any case, genetic algorithms are likely to provide a promising line of research in DNA computing for at least three reasons. First, GA's are based on populations of individuals. Second, they can make use of the noise that, at the present, is unavoidable in DNA computing. Third, because of the semantic match between GA's and the role of DNA in evolution, there already exists a large set of enzymes for manipulating DNA in ways likely to be useful for the implemention of GA's.

2.2.3 Breaking DES

More dramatic however, was Boneh et al.'s (Boneh et al., 1995a) description of an algorithm for breaking the Data Encryption Standard (DES). This is just a variation on the boolean circuit satisfiability algorithm. The only requirement for the algorithm to find a 56 bit DES key is one example of a 64 bit sample of plain text and the encryption of that text. The algorithm follows in three steps:

1. Generate all possible 56 bit keys in the standard "generate and isolate" style of DNA computation.

2. For all keys in parallel, simulate the behavior of the DES circuit given that key and the 64 bit sample of plain text. Section 2.1.1 gives a description of such a circuit simulation. The DES circuit requires only a minor elaboration to handle XOR gates and 64 entry lookup tables. The result of this is to append an encrypted version of the plain text to each strand of DNA representing a key.

3. Use the known encrypted version of the plain text to separate out all those keys that produce that encryption.

Boneh et al. use the somewhat more conservative estimate that 10^{17} strands of DNA can fit in a liter of water. Thus, assuming all operations work perfectly, including the generation of the initial keys, all the separation steps involved in simulating the circuit, and extracting the answer strands, such a process can break any encryption algorithm that uses keys less than or equal to 57 bits. One bit more than necessary in the case of DES. Boneh et al. claim the algorithm will work in the same amount of time for keys up to 64 bits long. This is predicated on a more optimistic estimate of the ability to manipulate 10^{20} strands of DNA and would probably require more than a liter of water.

Assuming that they can set up a laboratory that can do 32 separations in parallel, a table lookup can be done in one of these parallel separation steps. Their algorithm requires a total of 916 separation steps (128 table lookups, 788×3 XOR gates evaluated 3 at a time). With a rough estimate of 1 hour per separation step, and the calculation of 10 separation steps per day, DES could be broken in about 4 months of work. What is worse is that once this calculation is made, the results can be used repeatedly to locate other keys given the same pair of plain text and encrypted text under the new key. Needless to say, this is worth worrying about. However, before we run to the bank, let's focus on what it would take to carry out 916 separation steps successfully. That brings us to our next topic.

3 Getting it Wrong

Can errors be adequately controlled? Numerous authors (Adleman, 1994; Lipton, 1995; Roweis et al., 1996) have identified the problem of errors as the defining challenge for the success of DNA computation. We have only begun to comprehend the scale of this obstacle.

3.1 Getting it Wrong in Practice

Kaplan et al. (Kaplan et al., 1995) set out to replicate Adleman's original experiment (Adleman, 1994), and failed. Or to be more accurate, they state "At this time, we have carried out every step of Adleman's experiment, but we have not gotten an unambiguous final result." Adleman himself reported contamination of his 140-bp sample with 120-bp molecules. Kaplan et al. describe just how bad it gets. They used both a positive and a negative control. The positive control was a simple graph with in-degree 1 and out-degree 1 for every node between v_{in} and v_{out}. This formed a single (Hamiltonian path). The negative control was a graph identical to Adleman's except a single edge was removed making the HPP unsolvable.

Errors appeared at every stage. After annealing and ligation (step 1) gel electrophoresis showed a wide smear with some peaks for different lengths generated under all three experiments. They expected to see bands at the different "possible" path lengths. Even the positive control, with such a simple structure, resulted in many different length paths, including "impossible" paths longer than the total number of edges in the graph. When they cut out this path and ran it again on the gel, they found bands at 120 and 140-bp, suggesting either contamination due to the shorter (faster) strands getting caught in the gel, or "an unusual configuration of 3 strands of partially hybridized DNA." Kaplan et al. suggest that incomplete ligation, irregular ligation, heterogeneous sequence-dependent shapes of different paths, paths that break in the gel, temporary binding of proteins to the DNA in the gel, or single strands hanging off the ends may explain the smear. In other words, there are many potential problems in both ligation and gel electrophoresis that interfere with the computation. Furthermore, it is extremely hard to separate strands of similar lengths. After two cycles of electrophoresis and PCR, Adleman had to repeat the electrophoresis four times on the product of the third PCR (Kaplan et al., 1995), each time removing the 120-bp band in an attempt to reduce contamination.

Things only get worse with the introduction of PCR. The polymerase will extend any double strand that has a single 5' strand hanging off the end. In Adleman's experiment PCR immediately follows the ligation of the first step. All of the undesired paths and fragments of paths are still present in the solution. PCR using the source and destination nodes as primers is supposed to selectively amplify those paths starting with the source and ending with the destination nodes. However, any partially built paths would also be acted upon, although presumably only once. PCR is designed with the assumption that the proportion of primer is far greater than the proportion of template in the solution. If there are a lot of templates, the interactions between the templates can interfere with the PCR (Kaplan et al., 1995). In this case, the edge and node strands can certainly interact. Finally, they estimated that for each magnetic bead extraction in step 4, they only recovered approximately 50% of the DNA (Kaplan et al., 1995). None of this is encouraging.

No further work analyzing error rates has been published in the field. The polymerase enzymes in PCR do make mistakes when they are synthesizing the copies of the DNA strands. This is true even in the absence of incomplete ligation products. Depending on the polymerase enzyme used, the error can be as low as 7×10^{-7} per base pair (for *Pfu*) (Cha and Thilly, 1995). There is evidence that an alternative process, called the Ligase Chain Reaction (LCR), does better than most PCR enzymes (Dieffenbach and Dveksler, 1995). Sequencing DNA, that is reading out the order of the nucleotides in a strand, requires the use of both PCR and gel electrophoresis. Thus the problem of errors in the sequencing reduces to the problems of errors in PCR and gel electrophoresis. Furthermore, the specificity of annealing depends on many factors including temperature, salt content of the solution, and the proportion of G's and C's relative to T's and A's in the sequences. Finally, merging two test tubes can break DNA strands, and leave others stuck to the surface of the discarded tube.

3.2 Getting it Wrong in Theory

Because of the dominance of the "generate and test" technique in the field of DNA computing, people concerned with containing and controlling error rates have focused on the two important phases of that approach: generating the set of potential solutions and filtering out the correct solutions.

3.2.1 Covering the Solution Space

One common procedure in DNA computation that has generated some analysis is the process of generating all bit strings of length n. What is the chance that all 2^n are actually generated by the proposed random processes? Adleman argues that the quantity of DNA "should grow exponentially with the number of vertices in the graph."(Adleman, 1994) However, if in general we assume that solution sequences are being created at random, then the problem of how much DNA you need, or how many solutions you need to generate, reduces to the coupon collector's problem. This problem states that if you want to collect the entire set of n different coupons by randomly drawing a coupon each trial. You can expect to make $O(n \log n)$ drawings before you collect the entire set. Similarly, the amount of DNA required to create all 2^n potential solutions is actually $O(n2^n)$. This was first noted in (Linial and Linial, 1995). Bach et al. (Bach et al., 1996) show that Lipton's algorithm (Lipton, 1995) for generating random solutions reduces to the coupon collectors problem. Even though there is some bias in the process due to the equal amounts of "0" and "1" subsequences added to the test tube, the first 2^{n-1} solutions form in a relatively unbiased fashion. Thus, in order to have a reasonable chance $(1-\epsilon)$ of generating all possible solutions under Lipton's algorithm, you need to use $O(n2^n)$ DNA.

3.2.2 Separation by Subsequence

Most of the work up to the present has focused on the problem of the separation step (Amos et al., 1996; Boneh and Lipton, 1996; Karp et al., 1995; Roweis et al., 1996). This is not surprising given its prominence in DNA algorithms. The separation step is estimated to exclude a correct strand (a false negative) with .05 probability and include an incorrect strand (a false positive) with 10^{-6} probability (Karp et al., 1995; Amos et al., 1996). Thus, the chance of avoiding an error in the 916 successive separation steps of the DES breaking algorithm is vanishingly small. There have been three significant responses to this problem.

Boneh and Lipton propose using PCR intermittently to amplify the strands in the test tube (Boneh and Lipton, 1996). Over a series of separation steps, the proportion of "good to bad" strands should grow, and so regular amplification helps to make sure you do not lose the good strands in the wash. At the end of the process, one can sample the remaining DNA in the final test tube and check it on a traditional computer. It is usually trivial to check whether the encoded answer is actually a correct answer to the problem. However, if the algorithm is relatively simple, for example adding two binary numbers (Guarnieri et al., 1996), then checking the answer might take just as long as generating the answer on a silicon computer. Furthermore, this technique of repeated amplifications only works for a "decreasing volume" algorithm. That is, an algorithm where the number of distinct DNA strands in the test tube of interest decreases as the algorithm progresses.

Both Karp et al. (Karp et al., 1995) and Roweis et al. (Roweis et al., 1996) replace a single separation with a series of separations. They trade off time and space (number of test tubes) for accuracy. If the separation step divides a set of DNA strands into a "True" test tube and a "False" test tube, the simplest version of this is to repeat the separation step on the False test tube, in order to lower the rate of false negatives. A more sophisticated version of this is to do repeated separations on both results creating an array of test tubes $1, \ldots, n$, where test tube k contains all those strands that have been processed as "True" k times. Thus a DNA strand does a biased random walk across the array. Roweis et al. suggest fixing the size of the array and making absorbing boundaries out of the test tubes at the ends. They point out that any DNA strands that reach the boundary test tubes can immediately be used in the next step of the computation. This creates a form of pipelining. By increasing the size of the array and thus the number of separation steps that must be performed, the error rates can be made arbitrarily small.

Finally, Amos et al. (Amos et al., 1996) take the Captain Kirk approach, i.e., they change the rules of the game. They develop a framework for DNA computation that avoids the use of separation by subsequence. Or more accurately, they implement separation by subsequence not by annealing the subsequence and extracting using magnetic beads or a fixed column, but by a series of "mark, cut, and separate by length" operations. "Mark" is implemented by introducing the Watson-Crick complement of a sequence to select all those strands that do not contain the desired subsequence. In other words, the "False" strands are marked. Then a polymerase enzyme is added to the solution. This enzymes require a short double stranded section (the "primer") of DNA in order to begin their work of building the complement of the remaining single stranded portion. Polymerases only extend the primer strand in one direction, starting from the 3' end of the primer. Once the polymerases have finished their work, strands have been marked by making them double stranded DNA. Every bit position in these strands has been designed to include a subsequence that is recognized by a restriction enzyme. Amos et al. suggest using the *Sau*3AI enzyme which will only cut double stranded DNA. Thus all the marked strands are cut by the enzyme. Then using gel electrophoresis, the intact single strands at full length can be separated from the undesired fragments.

In order to use this method for algorithms that require the use of both the True and the False results of a separation, one has to copy the original test tube and then run a complementary modified separation procedure on the two test tubes. This will destroy the True strands in one test tube and the False strands in the other. Since they quote that restriction enzymes are "guaranteed" to cut any double stranded DNA containing the appropriate restriction site, Amos et al. claim this method is a significant improvement over methods that use separation by annealing. Unfortunately, they rely instead on the highly error-prone process of gel electrophoresis (Kaplan et al., 1995).

The problem of errors remains the largest area of open questions at present. Biological processes are messy almost by definition. More work needs to be done to design algorithms for handling the errors in the processes. Following Amos et al. (Amos et al., 1996) we might also look into models of computation that exploit relatively error-free, and practical, biological processes. Most importantly, more experiments need to be carried out in the lab to determine what processes are in fact practical for computation and to measure the resulting error rates.

4 Getting Wet

The most disappointing thing about the field to date is the lack of experimental results. Besides Adleman's original work (Adleman, 1994), and Kaplan et al.'s attempted replication (Kaplan et al., 1995), few have actually ventured into the lab. Oghihara and Ray (Ogihara and Ray, 1996) provide one harbinger of doom. Two out of three attempts to evaluate a 3-gate circuit failed due to "pipetting error." (Ogihara and Ray, 1996)

4.1 Computation in the Plane

One exception to the dearth of experimental results is the work by Winfree et al. (Winfree et al., 1996). They have made a start at implementing the Blocked Cellular Automata (BCA). So far, they have provided some initial evidence that they can construct a planar rectangle of DNA and that these rectangles seem to anneal to each other as desired. However, the good stuff, an actual computation, is yet to come. It is also not clear that a BCA can be made into an efficient engine for computation. In particular, a single annealing error destroys a BCA computation (Winfree et al., 1996).

Another exception is the contributions of an interdisciplinary group at the University of Wisconsin. They propose the idea of carrying out DNA computation on a surface rather than in solution (Cai et al., 1996; Liu et al., 1996). This approach sacrifices quantity of DNA strands for ease of manipulation. One end of each DNA strand is attached to a planar surface. In their experiments they have attached the 5' end to a glass surface. They estimate that they can fit 10^{12} strands on 1 cm^2. Instead of separating sequences into physically different containers, they operate on subsets of the strands by marking strands and then operating either on the marked or unmarked strands. This has the benefit of not losing strands in a separation operation. Similar to Amos et al. (Amos et al., 1996), they "mark" a strand by making it a double strand through the process of annealing a primer to it and then using a polymerase to build the complementary strand. The primer here can just be a subsequence that is being used to select strands. Unmarking is then just a matter of melting off the complementary strands. They can destroy a marked or an unmarked strand through the introduction of an exonuclease. This is an enzyme that chews up double or single (depending on the enzyme) stranded DNA, breaking the covalent bonds between the nucleotides. The remaining important operation in their model of DNA computation is the appending of a new subsequence to the marked (or unmarked) strands. Depending on the polymerase enzyme used to mark the strands, the end of a strand is either "blunt" or has a one base (A) "sticky" end. They ligate the new subsequence to the marked strands using a ligase that only works on double stranded DNA. Ligation to sticky ended DNA is "substantially more efficient" (Cai et al., 1996).

Liu et al. have begun using the surface based approach to solve a 5 variable instance of the circuit SAT problem (Liu et al., 1996). Their representation of the state of each variable is encoded in an extremely dense fashion. A single nucleotide codes each variable's state. These 5 nucleotides are sandwiched between 20-mer primers for PCR. They have attempted three preliminary experiments to see if their approach is feasible. First they examined the specificity of the "mark" operation. They attached two different strands to different patches of the glass surface and then allowed fluorescently tagged complementary strands to anneal to the different strands. However, one of the two attached strands had a 1-base mismatch with the annealing strands. While they found dramatically higher rates of attachment in the perfectly matching strands, they did detect some fluorescence in the patch with the 1-base mismatch strands. Evidences suggests this is due to the tagged strands becoming attached to the surface itself, and not to them annealing to the 1-base mismatched strands. Specifically, washing in distilled water, which should break apart the double strands, still leaves some fluorescence in the patch. Liu et al. are currently working on eliminating this problem through using alternative surface chemistries.

Their other two experiments were entirely successful. They showed that treatment with Exonuclease I, which specifically degrades only single stranded DNA, effectively implements the "destroy unmarked" operation.[9] They also found that they could combine the "mark"

[9]Exonuclease I also works in a solution, but on a surface it requires Bovine Serum Albumin (BSA) or non-ionic surfactant Tritron X-100 to facilitate its activity

and “destroy unmarked” operations successfully. However, it should be noted that they did not try to selectively mark sequences. Rather, for 4 different patches they simply marked all the strands, while 4 other patches were left unmarked. Then all 8 patches were treated with Exonuclease I, and only the unmarked patches were destroyed.

Several challenges remain. The “destroy marked” and the “append” operations have yet to be demonstrated. “Append” in particular is required in order to implement the Boneh et al. algorithm for boolean circuit satisfiability (Boneh et al., 1995b). At present they use an attachment chemistry that attaches the strand at its 5’ end. This will not work for the operation of marking strands by their inclusion of a subsequence. In order to mark a subsequence, the complement of the subsequence is allowed to anneal to strands on the surface. Then a polymerase is added which builds out the rest of the complementary strands based on the presence of that primer. However, polymerases build towards the 5’ end of the complementary strand. So, in order to make the exposed end double stranded, and so avoid degradation from Exonuclease I, it is necessary to attach the 3’ end of the strand to the surface and leave the 5’ end free.

As was mentioned, the specificity of the “mark” procedure needs refinement. Furthermore, as the base strands get longer, it becomes harder to discriminate single base mismatches. In their experiments, Liu et al. use a 15-mer, 5 nucleotides to encode the 5 variables and 10 left over for future demonstrations on more complex problems. They suggest that for problems requiring many more nucleotides, the operations might have to be divided up into 15 or 20-mer subsequence “words” in order to attain single base pair mismatch levels of discrimination. Even if this works, annealing is not 100% efficient. So we effectively land right back with the problem of false negatives and positives that plagued the “separation by subsequence” operation.

4.2 Adding a Further Complication

The most recent published contribution has been a strikingly new form of “one-pot” computation. Guarnieri et al. (Guarnieri et al., 1996) have developed a procedure to implement the addition of two non-negative binary numbers through a chain reaction. Furthermore, they have demonstrated the algorithm on pairs of single bit binary numbers. They implement an addition through a cascade of branches in the chain reaction which are predicated on the value of the sum in the previous bit position. I have summarized their algorithm in pseudo-C code below.

```
int number1[n]; /*Binary numbers represented as arrays of bits.*/
int number2[n];

carry = 0; /*Keeps track of the carryover from the previous bit.*/
n2 = 0;    /*An intermediary variable to hold carry + number2.*/

for (i=0; i < n; i++) {

   if (number2[i] == 0) /*Calculate number2 + carry.*/
      if (carry == 0)
```

```
         n2 = 0;
      else n2 = 1;
   else
      if (carry == 0)
         n2 = 1;
      else n2 = 2;

   if (number1[i] == 0 /*Calculate number1 + n2.*/
      switch (n2) {
         case 0: output(0); carry = 0; break;
         case 1: output(1); carry = 0; break;
         case 2: output(0); carry = 1; break;
      }
   else
      switch (n2) {
         case 0: output(1); carry = 0; break;
         case 1: output(0); carry = 1; break;
         case 2: output(1); carry = 1; break;
      }
}

if (carry == 1) output(1); /*Spit out any carryover.*/
```

Addition of 11 + 11: bit 0

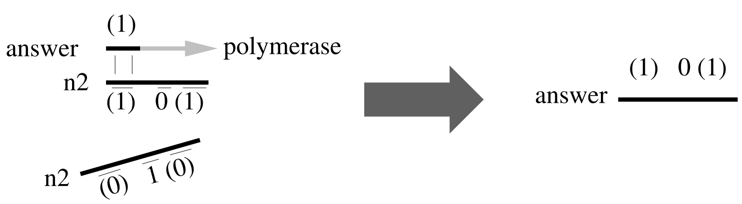

Figure 7: Addition proceeds from least significant to most significant bit. The numbers in parentheses prepresent subsequences encoding partial sums at each bit position. The numbers without parentheses are the bits of the answer. The subsequences are all assumed to include patterns that label the bit position, so that (0) at bit position 0 is a different sequence than (0) at bit position 1. Multiple fragments represent the operands n1 and n2. The addition for bit position 0 is simpler than addition for the other positions because there is no carry bit from a previous computation. Thus at the start, least significant bit of the "answer" is the same as the least significant bit of n1. This bit is then added to the least significant bit of n2 and the result includes both the least significant bit of the sum of n1 and n2 as well as a pattern representing the presence (or absence) of a carry-over bit.

Notice that there is no explicit addition operation in the above algorithm, besides my encoding of the iteration. The branching statements are implemented as annealing operations

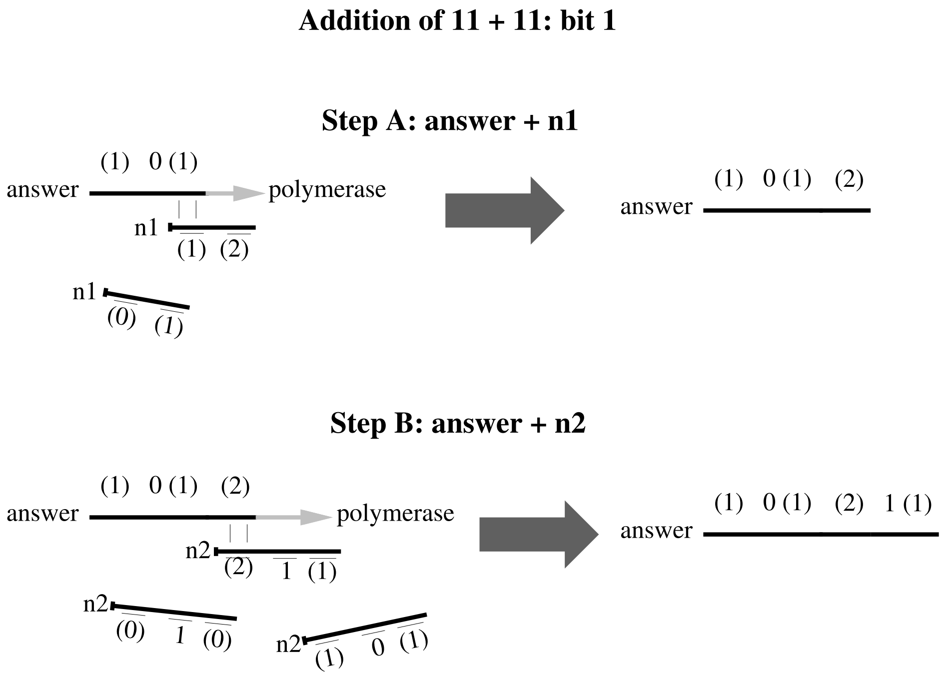

Figure 8: For all bit positions other than the least significant bit, addition has to follow in two steps. In step A, the carry-over bit is added to the bit of n1. Then, in step B, the result is added to the bit of n2. Note again that n1 and n2 are both represented as a set of DNA fragments, only one of which will bind to the answer strand in the chain reaction. The 3' ends of these fragments have been altered to prevent the polymerase extending them.

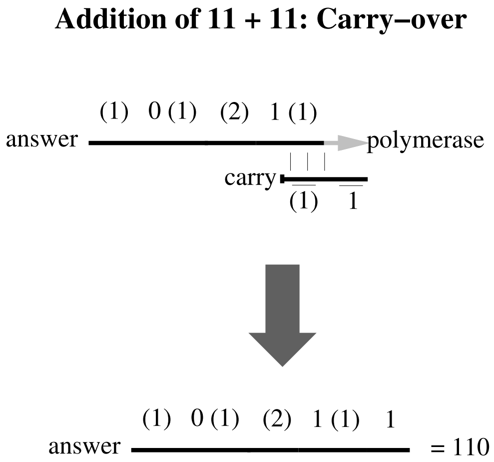

Figure 9: Finally, a strand representing the carry-over bit anneals to the answer and completes the process of constructing the answer: 110 (in reverse order), or 6 in binary.

between the number being constructed (the sum of the two input numbers) and a set of possible complements. Once the number being built (the sum) anneals to the appropriate strand, a polymerase enzyme extends the sum. Then the two strands are melted apart and then cooled so that the sum is then free to anneal to the next bit position value in the sequence. Each value in each bit position has a unique sequence. Thus, if the `number2` had a 1 in the 2^3 position, there would be two strands added to the soup. One representing the code `if (carry == 0) n2 = 1` and one representing `if (carry == 1) n2 = 2`. Then, if the sum had a string at its 3' end representing the complement $\overline{\texttt{carry == 1}}$ in the 2^2 position, it would bind to the second option and polymerase would add on the string representing `n2 = 2` at the 2^3 position. The same process follows for the reaction with the 2^3 position in `number1` but this time, three strings would be present in the test tube representing how to extend the sum under each of the three cases `n2 = 0, 1, 2`. In this way, the addition operation is actually built into the encoding of the two binary numbers. At the end of the chain reaction, the resulting sum must be sequenced in order to determine the result. The addition of $11 + 11$ ($3 + 3$ in binary) is depicted in Figures 7, 8, and 9.The amount of DNA necessary to implement an n-bit addition is only O(n), although it requires the construction of 7 unique subsequences (and their complements) for each bit position.

Guarnieri et al. point out the worst flaw in this algorithm. The encoding of the output number is different from the encoding of the two input numbers. Thus, the output cannot be fed back into the DNA computer for future computation without translating it back into the input format. There are further limitations. As they state from the beginning, this procedure only works for nonnegative numbers. Theoretically, this algorithm should be able to handle different addition operations being carried out in parallel as long as all subsequences representing different bit position branches in the test tube are unique and unlikely to bind to each other. Although it is a "one-pot" computation,[10] it must be driven by temperature cycling. The speed of the cycling is limited in turn by the speed of the annealing reaction and the melting apart of the strands. Since the strand representing the sum may well anneal back again to the last strand that allowed the polymerase to extend the sum, we are not guaranteed that the addition operation will make progress in every temperature cycle. In fact, the strand that will allow progress has fewer base pairs that match the sum than the last strand that matched it.

Polymerase enzymes make mistakes with frequencies of between 10^{-4} and 10^{-7} per base pair depending on the enzyme used (Cha and Thilly, 1995). The algorithm could probably handle some amount of noise since errors would not cascade as long as they did not prevent the next annealing of the appropriate strands. However, the gel electrophoresis used to examine the results of their one bit addition experiments showed a large amount of noise. They did not sequence the results, since the lengths served as indicators that at least something of the right size had been built. However, the frequencies of errors in their system is certainly non-zero and has yet to be ascertained. Guarnieri et al. suggest the use of redundant steps, increased hybridization stringency (making the differences between the subsequences greater), and occasional isolation and amplification of the sum strand under construction in

[10]It is a "one-pot" algorithm in theory. In implementation, Guarnieri et al. added the terminal "carry" strand to the solution after they had carried out the addition on the first bit position of 1+1. So it still remains to be demonstrated how effective a chain reaction will be.

order to reduce errors.

5 The Promise

> "Despite these encouraging theoretical advances, we emphasize that substantial engineering challenges remain at almost all stages and that the ultimate success or failure of DNA computing will certainly depend on whether these challenges can be met in laboratory investigations." (Roweis et al., 1996)

This is the most important observation of the state of DNA computing. Talk is cheap, but bench work is hard. It is perhaps reminiscent of the early days of electronic computers. The number of theoretical models far out number the number of biochemical experiments in DNA computation. It is clear that the first DNA computer will have to be something akin to the ENIAC. That is, it will have to manipulate an unprecedented number of components and allow for high levels of noise and failure. While there are no obviously insurmountable problems at this point, I expect that there will be a good deal more bad news about the error rates in our manipulations before we make much progress on building such a DNA computer.

Adleman made specific claims about the speed, energy requirements and information density of DNA computers. These are claims about the potential of DNA computing. In the two years that have followed Adleman's first experiment, no one has yet managed to build a simple DNA computer or even a prototype. With the vast majority of engineering hurdles looming tall on the horizon, it would be premature to try to assess these claims. However, a few cautions are in order. The speed of DNA computing depends on the massive parallelism inherent in the medium. If an algorithm cannot be transformed to exploit this parallelism, it will not run quickly on a DNA computer. Most of the proposed models require far more energy than the annealing reaction. Lo et al. point out that there are "enormous energy inefficiencies" in electrophoresis, the thermal cycling of PCR, and other manipulations of the DNA (Lo et al., 1995). While the molecular interactions of the DNA are likely to require little energy, at the very least the I/O is going cost us (Schneider, 1994; Winfree et al., 1996). Finally, the density of information will be limited by the need for redundancy to overcome some forms of errors. It may be more practical to move to surface based chemistries (Liu et al., 1996; Cai et al., 1996) which dramatically reduces the density of information in exchange for ease of manipulation.

5.1 The Next Steps

At this point, it isn't even clear what all the engineering challenges are. There are, however, plenty of problems to work on. Perhaps the most pressing problem is the need to ascertain and address the error rates for the different manipulations. This bears on the more general problem of what model of computation should we use. The question is no longer, can we do computation with DNA? But rather, how should we do computation with DNA? To settle on an appropriate model will require both a theoretical and a practical contribution. The model must be a compromise between what we can implement biologically with tolerable levels of errors and what we can use to efficiently execute interesting algorithms.

5.1.1 Theory

The overarching theoretical open problem is what model of universal computation will work best. But beneath this, there are a number of smaller open problems. Given the variety of possible operations for manipulating DNA, with their associated error rates under current technology, what are the best operations to use. This is not simply a matter of choosing the operations with the lowest error rates. The choice depends on the algorithms that a set of manipulations can efficiently implement as well as the design of algorithms for trading off time and space for accuracy. For example, the H-systems rely only upon annealing and cutting. No work has been published analyzing the effect of errors in these systems along with algorithms for containing such errors. In general, Gifford (Gifford, 1994) suggests the need to design processes that prevent creation of false positives.

Another familiar challenge is the design of "one-pot" computational models. Implementing the computation as a chain reaction in a single large test tube is not desirable in and of itself. It is merely a proxy for the goals of automation, speed, and efficiency. Furthermore, it would be helpful to continue the search for useful problems that can be solved efficiently with DNA computation.

DNA algorithms depend on the precise manipulation of DNA sequences. For these algorithms to work, the sequences must conform to a rigorous set of restrictions (Roweis et al., 1996). We need to develop implementable algorithms for generating subsequences such that a combination of subsequences is

1. Unique
2. Does not significantly bind to itself. For every subsequence x of length $|x| > n$, there should be no subsequences , or subsequences similar to $\overline{x}$, anywhere in the sequence.
3. Easily identified by any of it constituent subsequences. This means that for any given subsequence s there are no other subsequences anywhere in the strand with more than k bases matching s, where k is a parameter.

Not to mention that sequences should be short enough to be cost effective, easily meltable into single strands, and unlikely to break under the stresses of the manipulations.

The search continues for a DNA computing application that is both practically implementable and more efficient than the solution in silicon computation. That is, we are still looking for a successful niche for DNA computing.

5.1.2 Practice

There are so many open problems in the practice of DNA computing that it is hard to know where to begin. In general, what does it take to implement a universal DNA computer? Can we make a "one-pot" DNA computer (Amenyo, 1996)? But before that is answered, we (Boneh et al., 1995b) can ask, what other operations are possible? And what are their trade offs? We know that long strands of DNA tend to break. What are the limitations on strand length? And what environmental factors influence those limitations?

If there is one question that bars significant progress it is what are the error rates for the various operations proposed in the theoretical models? This must be addressed under conditions of massive parallelism. How can we deal with the error rates? These are the most daunting current problems (Adleman, 1994; Lipton, 1995; Roweis et al., 1996).

Molecular biology is a large and quickly progressing field. Little work has been done to mine it for useful DNA computational tools. What other technologies might be useful? Recent work has shown that DNA with modified backbones, PNA (Egholm et al., 1993; Ito et al., 1992) and DNG (Dempcy et al., 1995), can bind to DNA and the resulting double strands are more stable than the natural DNA-DNA strands. PNA binds so strongly, it can displace one of the strands in a DNA-DNA double strand (Nielsen et al., 1991). This is all tantalizing, but it only scratches the surface of advances in molecular biology. As a caution, Kaplan et al. point out that most biological manipulations are designed to be very narrowly applicable, often with implicit filtering steps (Kaplan et al., 1995). For example, when error-ridden products are inserted into bacteria they are generally not incorporated or expressed. So the biologists can tolerate a high degree of errors in their manipulations. What can be done to adapt the biological manipulations to the requirements of computation?

On the administrative side, to what extent can the operations be automated? And, what is the monetary expense of the different models? Little reference has been made to the cost of the materials or the procedures. Kaplan et al. state that the price of synthetic DNA is falling rapidly. Writing in July of 1995, they claim that the price had fallen by 50% in the previous year.

Are there faster ways to determine the sequence of a strand of DNA? Answering this question will be key to the actual use of DNA computers. As it stands, I/O is a laborious process. There is some evidence that different sequences have different conductances (Paterson, 1995). This may form the basis for a new technique for sequencing DNA.

Finally, is there a better molecular substrate for computation than DNA (Adleman, 1996)? The above questions can be asked for any molecule, and the answers will largely determine progress in the field.

5.2 Computation in the Eye of the Beholder

In the same issue of Science where Adleman's experiment first appeared, Gifford (Gifford, 1994) expressed the hope that "we will identify a naturally occuring universal computational system." Might there be computation going on in the eye of the beholder? Unfortunately, we are a long way from recognizing such a system, if it exists. Meanwhile, one cause for real optimism in the future of DNA computing is its synergy with molecular biology. Almost any advances in techniques in one field will be useful to the other. Thus, even if DNA computers never manage to leap-frog traditional silicon computers, they are still bound to be of value.

Is DNA computing doomed to uselessness? This seems unlikely. None of the obvious technical challenges seem overwhelming. Even if it proves impossible to implement a universal DNA computer able to run all our software at blazingly fast speeds, the massively parallel nature of DNA computing is sure to be good for some practical problems, as was illustrated in the breaking of DES (Boneh et al., 1995a). Will DNA (or a similar form of molecular substrate) replace silicon as our computational medium of choice? This is harder to answer from our current perspective. It depends on solving the problems associated with implementing a universal computing machine in DNA. We know this is a theoretical possibility. What is more, it depends on solving these problems before advances in silicon or other forms of computing outstrip the potential of the DNA substrate. The feasibility of overcoming the error rates remains unclear. There is one thing of which we can be sure, there is going to be a lot more toner melted over this topic.

References

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

Adleman, L. M. (1996). On constructing a molecular computer. In (Baum and Lipton, 1996). Based on Manuscript, Computer Science Department, University of Southern California, January 11, 1995.

Amenyo, J.-T. (1996). Mesoscopic computer engineering: Automating DNA-based molecular computing via traditional practices of parallel computer architecture design. In (AMS96, 1996).

Amos, M., Gibbons, A., and Hodgson, D. (1996). Error-resistant implementation of DNA computations. In (AMS96, 1996). Previously: Research Report CS-RR-298, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England, January 1996.

AMS96 (1996). *Proceedings of the Second Annual Meeting on DNA Based Computers, held at Princeton University, June 10-12, 1996.*, DIMACS: Series in Discrete Mathematics and Theoretical Computer Science., unknown. American Mathematical Society, unknown. To appear.

Bach, E., Condon, A., Glaser, E., and Tanguay, C. (1996). *DNA Models and Algorithms for NP-complete Problems*, pages 290–299. IEEE Computer Society Press, New York, NY.

Baum, E. B. (1995). Building an associative memory vastly larger than the brain. *Science*, 268:583–585.

Baum, E. B. and Lipton, R. J., editors (1996). *DNA Based Computers*. Number 27 in DIMACS: Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society. Proceedings of the DIMACS workshop. Held Apr. 4, 1995 at Princeton University.

Beaudry, A. A. and Joyce, G. F. (1992). Directed evolution of an rna enzyme. *Science*, 257:635–641.

Beaver, D. (1995). Molecular computing. Technical Report TR95-001, Penn State University.

Blumberg, A. J. (1996a). General purpose parallel computation on a dna substrate. Technical Report AIM-1589, MIT Artificial Intelligence Laboratory.

Blumberg, A. J. (1996b). Parallel function application on a dna substrate. Technical Report AIM-1588, MIT Artificial Intelligence Laboratory.

Boneh, D., Dunworth, C., and Lipton, R. J. (1995a). Breaking DES using a molecular computer. Technical Report CS-TR-489-95, Princeton University.

Boneh, D., Dunworth, C., and Sgall, J. (1995b). On the computational power of DNA. Technical Report TR-499-95, Princeton University.

Boneh, D. and Lipton, R. J. (1996). Making DNA computers error resistant. In (AMS96, 1996). Previously: Princeton University TR-491-95, May 1995

Brown, T. A. (1995). *Gene Cloning*. Chapman & Hall, London.

Cai, W., Condon, A. E., Corn, R. M., Glaser, E., Fei, Z., Frutos, T., Guo, Z., Lagally, M. G., Liu, Q., Smith, L. M., and Thiel, A. (1996). The power of surface-based DNA computation. Preprint.

Cha, R. S. and Thilly, W. G. (1995). Specificity, efficiency, and fidelity of PCR. In Dieffenbach, C. W. and Dveksler, G. S., editors, *PCR Primer*. Cold Spring Harbor Laboratory Press, Plainview, NY.

Csuhaj-Varjú, E., Freund, R., Kari, L., and Păun, G. (1996). DNA computation based on splicing: universality results. In Hunter, L. and Klein, T., editors, *Biocomputing: Proceedings of the 1996 Pacific Symposium*, Singapore. World Scientific Publishing Co., Singapore. Also: Technical report 185-2/FR-2/95, TU Wien, Institute for Computer Languages, Wien, Austria, 1995, http://www.csd.uwo.ca/ lila/four.ps.

Dempcy, R. O., Browne, K. A., and Bruice, T. C. (1995). Synthesis of a thymidyl pentamer of deoxyribonucleic guanidine and binding studies with DNA homopolynucleotides. *Proceedings of the National Academy of Science, USA*, 92:6097–6101.

Dieffenbach, C. W. and Dveksler, G. S. (1995). *PCR Primer*. Cold Spring Harbor Laboratory Press, Plainview, NY.

Egholm, M., Buchardt, O., Christensen, L., Behrens, C., Freler, S. M., Driver, D. A., Berg, R. H., Kim, S. K., Norden, B., and Nielsen, P. E. (1993). PNA hybridizes to complementary oligonucleotides obeying the watson-crick hydrogen-bonding rules. *Nature*, 365:566–568.

Gannon, F. and Powell, R. (1991). Construction of recombinant DNA. In Brown, T. A., editor, *Essential Molecular Biology*. Oxford University Press, Oxford.

Gifford, D. K. (1994). On the path to computation with DNA. *Science*, 266:993–994.

Goldberg, D. E. (1989). *Genetic Algorithms: In Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Guarnieri, F., Fliss, M., and Bancroft, C. (1996). Making DNA add. *Science*, 273(5272):220–223.

Hartmanis, J. (1995). On the weight of computations. *Bulletin of the European Association for Theoretical Computer Science*, 55:136–138.

Head, T. (1987). Formal language theory and DNA: an analysis of the generative capacity of specific recombinant behaviors. *Bulletin of Mathematical Biology*, 49(6):737–759.

Ito, T., Smith, C. L., and Cantor, C. R. (1992). Sequence-specific DNA purification by triplex affinity capture. *Proceedings of the National Academy of Science, USA*, 89:495–498.

Kaplan, P. D., Cecchi, G., and Libchaber, A. (1995). Molecular computation: Adleman's experiment repeated. Technical report, NEC Research Institute.

Karp, R., Kenyon, C., and Waarts, O. (1995). Error resilient DNA computation. Research report 95-20, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, 46, Allée d'Italie 69364 LYON CEDEX 07 - FRANCE.

Linial, M. and Linial, N. (1995). Letters to science. *Science*, 268:481.

Lipton, R. J. (1995). DNA solution of hard computational problems. *Science*, 268:542–545.

Liu, Q., Guo, Z., Condon, A. E., Corn, R. M., Lagally, M. G., and Smith, L. M. (1996). A surface-based approach to DNA computation. In (AMS96, 1996).

Lo, Y.-M. D., Yiu, K. F. C., and Wong, S. L. (1995). Letters to science. *Science*, 268:481.

Lönneborg, A., Sharma, P., and Stougaard, P. (1995). Construction of a subtractive cDNA library using magnetic beads and PCR. In Dieffenbach, C. W. and Dveksler, G. S., editors, *PCR Primer*. Cold Spring Harbor Laboratory Press, Plainview, NY.

Lorsch, J. R. and Szostak, J. W. (1994). *In Vitro* evolution of new ribozymes with polynucleotide kinase activity. *Nature*, 371:31–36.

Nelson, M., Raschke, E., and McClelland, M. (1993). Effect of site-specific methylation on restriction endonucleases and DNA modification methyltransferases. *Nucleic Acids Research*, 21(13):3139–3154.

Nielsen, P. E., Egholm, M., Berg, R. H., and Buchardt, O. (1991). Sequence-selective recognition of DNA by strand displacement with thymine-substituted polyamide. *Science*, 254:1497–1500.

Ogihara, M. and Ray, A. (1996). Simulating boolean circuits on a DNA computer. Technical Report TR 631, University of Rochester, Computer Science Department.

Paterson, D. (1995). Electric genes. *Scientific American*, pages 33–34.

Roberts, R. J. and Macelis, D. (1993). REBASE—restriction enzymes and methylases. *Nucleic Acids Research*, 21(13):3125–3137.

Rothemund, P. W. K. (1996). A DNA and restriction enzyme implementation of Turing machines. In (Baum and Lipton, 1996).

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N. V., Goodman, M. F., Rothemund, P. W. K., and Adleman, L. M. (1996). A sticker based architecture for DNA computation. In (AMS96, 1996).

Sassanfar, M. and Szostak, J. W. (1993). An RNA motif that binds ATP. *Nature*, 364:550–553.

Schneider, T. D. (1994). Sequence logos, machine/channel capacity, Maxwell's demon, and molecular computers: a review of the theory of molecular machines. *Nanotechnology*, 5:1–18.

Stemmer, W. P. C. (1994). Rapid evolution of a protein *in vitro* by dna shuffling. *Nature*, 370:389–391.

Vahey, M. T., Wong, M. T., and Michael, N. L. (1995). A standard PCR protocol: Rapid isolation of DNA and PCR assay for β-globin. In Dieffenbach, C. W. and Dveksler, G. S., editors, *PCR Primer*. Cold Spring Harbor Laboratory Press, Plainview, NY.

Winfree, E. (1995). On the computational power of DNA annealing and ligation. In (Baum and Lipton, 1996).

Winfree, E., Yang, X., and Seeman, N. C. (1996). Universal computation via self-assembly of DNA: Some theory and experiments. In (AMS96, 1996). Draft.