DNA Computing

MARTYN AMOS

Manchester Metropolitan University, United Kingdom

Invited article for the Encyclopedia of Complexity and System Science, Springer, 2008

Article Outline

Glossary

DNA	Deoxyribonucleic acid. Molecule that encodes the genetic information of cellular organisms.
Enzyme	Protein that catalyzes a biochemical reaction.
Nanotechnology	Branch of science and engineering dedicated to the construction of artifacts and devices at the nanometre scale.
RNA	Ribonucleic acid. Molecule similar to DNA, which helps in the conversion of genetic information to proteins.
Satisfiability (SAT)	Problem in complexity theory. An instance of the problem is defined by a Boolean expression with a number of variables, and the problem is to identify a set of variable assignments that makes the whole expression true.

I Definition of the Subject and Its Importance

DNA computing (or, more generally, biomolecular computing) is a relatively new field of study that is concerned with the use of biological molecules as fundamental components of computing devices. It draws on concepts and expertise from fields as diverse as chemistry, computer science, molecular biology, physics and mathematics. Although its theoretical history dates back to the late 1950s, the notion of computing with molecules was only physically realised in 1994, when Leonard Adleman demonstrated in the laboratory the solution of a small instance of a well-known problem in combinatorics using standard tools of molecular biology. Since this initial experiment, interest in DNA computing has increased dramatically, and it is now a well-established area of research. As we expand our understanding of how biological and chemical systems process information, opportunities arise for new applications of molecular devices in bioinformatics, nanotechnology, engineering, the life sciences and medicine.

II Introduction

In the late 1950s, the physicist Richard Feynman first proposed the idea of using living cells and molecular complexes to construct "sub-microscopic computers." In his famous talk *"There's Plenty of Room at the Bottom"* [18], Feynman discussed the problem of "manipulating and controlling things on a small scale", thus founding the field of nanotechnology. Although he concentrated mainly on information storage and molecular manipulation, Feynman highlighted the potential for biological systems to act as small-scale information processors:

> The biological example of writing information on a small scale has inspired me to think of something that should be possible. Biology is not simply writing information; it is doing something about it. A biological system can be exceedingly small. Many of the cells are very tiny, but they are very active; they manufacture various substances; they walk around; they wiggle; and they do all kinds of marvelous things – all on a very small scale. Also, they store information. Consider the possibility that we too can make a thing very small which does what we want – that we can manufacture an object that maneuvers at that level! [18].

I Early Work

Since the presentation of Feynman's vision there there has been an steady growth of interest in performing computations at a molecular level. In 1982, Charles Bennett [8] proposed the concept of a "Brownian computer" based around the principle of reactant molecules touching, reacting, and effecting state transitions due to their random Brownian motion. Bennett developed this idea by suggesting that a Brownian Turing Machine could be built from a macromolecule such as RNA. "Hypothetical enzymes", one for each transition rule, catalyze reactions between the RNA and chemicals in its environment, transforming the RNA into its logical successor.

In the same year, Conrad and Liberman developed this idea further in [15], in which the authors describe parallels between physical and computational processes (for example, biochemical reactions being employed to implement basic switching circuits). They introduce the concept of molecular level "word processing" by describing it in terms of transcription and translation of DNA, RNA processing, and genetic regulation. However, the paper lacks a detailed description of the biological mechanisms highlighted and their relationship with "traditional" computing. As the authors themselves acknowledge, "our aspiration is not to provide definitive answers ... but rather to show that a number of seemingly disparate questions must be connected to each other in a fundamental way." [15]

In [14], Conrad expanded on this work, showing how the information processing capabilities of organic molecules may, in theory, be used in place of digital switching components. Particular enzymes may alter the three-dimensional structure (or *conformation*) of other *substrate* molecules. In doing so, the enzyme switches the *state* of the substrate from one to another. The notion of *conformational computing* (q.v.) suggests the possibility of a potentially rich and powerful computational architecture. Following on from the work of Conrad *et al.*, Arkin and Ross show how various logic gates may be constructed using the computational properties of enzymatic reaction mechanisms [5] (see Dennis Bray's article [10] for a review of this work). In [10], Bray also describes work [23, 24] showing how chemical "neurons" may be constructed to form the building blocks of logic gates.

II Motivation

We have made huge advances in machine miniaturization since the days of room-sized computers, and yet the underlying computational framework (the *von Neumann architecture*) has remained constant. Today's supercomputers still employ the kind of sequential logic used by the mechanical "dinosaurs" of the 1940s [13].

There exist two main barriers to the continued development of "traditional", silicon-based computers using the von Neumann architecture. One is inherent to the machine architecture, and the other is imposed by the nature of the underlying *computational substrate*. A computational substrate may be defined as *"a physical substance acted upon by the implementation of a computational architecture."* Before the invention of silicon integrated circuits, the underlying substrates were bulky and unreliable. Of course, advances in miniaturization have led to incredible increases in processor speed and memory access time. However, there is a limit to how far this miniaturization can go. Eventually "chip" fabrication will hit a wall imposed by the *Heisenberg Uncertainty Principle* (HUP). When chips are so small that they are composed of components a few atoms across, quantum effects cause interference. The HUP states that the act of observing these components affects their behavior. As a consequence, it becomes impossible to know the exact state of a component without fundamentally changing its state.

The second limitation is known as the *von Neumann bottleneck*. This is imposed by the need for the central processing unit (CPU) to transfer instructions and data to and from the main memory. The route between the CPU and memory may be visualized as a two-way road connecting two towns. When the number of cars moving between towns is relatively small, traffic moves quickly. However, when the number of cars grows, the traffic slows down, and may even grind to a complete standstill. If we think of the cars as units of information passing between the CPU and memory, the analogy is complete. Most computation consists of the CPU fetching from memory and then executing one instruction after another (after also fetching any data required). Often, the execution of an instruction requires the storage of a result in memory. Thus, the speed at which data can be transferred between the CPU and memory is a limiting factor on the speed of the whole computer.

Some researchers are now looking beyond these boundaries and are investigating entirely new computational architectures and substrates. These developments include *quantum computing* (q.v.), *optical computing* (q.v.), *nanocomputers* (q.v.) and bio-molecular computers. In 1994, interest in molecular computing intensified with the first report of a successful non-trivial molecular computation. Leonard Adleman of the University of Southern California effectively founded the field of DNA computing by describing his technique for performing a massively-parallel random search using strands of DNA [1]. In what follows we give an in-depth description of Adleman's seminal experiment, before describing how the field has evolved in the years that followed. First, though, we must examine more closely the structure of the DNA molecule in order to understand its suitability as a computational substrate.

III The DNA Molecule

Ever since ancient Greek times, man has suspected that the features of one generation are passed on to the next. It was not until Mendel's work on garden peas was recognized [39] that scientists accepted that both parents contribute material that determines the characteristics of their offspring. In the early 20^{th} century, it was discovered that *chromosomes* make up this material. Chemical analysis of chromosomes revealed that they are composed of both *protein* and *deoxyribonucleic acid*, or *DNA*. The question was, which substance carries the genetic information? For many years, scientists favored protein, because of its greater complexity relative to that of DNA. Nobody believed that a molecule as simple as DNA, composed of only four subunits (compared to 20 for protein), could carry complex genetic information.

It was not until the early 1950s that most biologists accepted the evidence showing that it is in fact DNA that carries the genetic code. However, the physical structure of the molecule and the hereditary mechanism was still far from clear.

In 1951, the biologist James Watson moved to Cambridge to work with a physicist, Francis Crick. Using data collected by Rosalind Franklin and Maurice Wilkins at King's College, London, they began to decipher the structure of DNA. They worked with models made out of wire and sheet metal in an attempt to construct something that fitted the available data. Once satisfied with their double helix model, they published the paper [43] (also see [42]) that would eventually earn them (and Wilkins) the Nobel Prize for Physiology or Medicine in 1962.

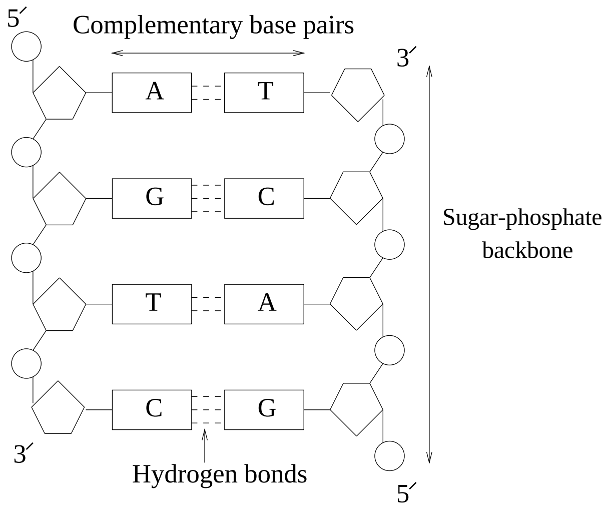

Figure 1: Structure of double-stranded DNA (from [3]).

I DNA Structure

DNA (deoxyribonucleic acid) [44] encodes the genetic information of cellular organisms. It consists of *polymer chains*, commonly referred to as DNA *strands*. Each strand may be viewed as a chain of *nucleotides*, or *bases*, attached to a sugar-phosphate "backbone." An n-letter sequence of consecutive bases is known as an n-mer or an *oligonucleotide* (commonly abbreviated to "oligo") of length n. Strand lengths are measured in *base pairs* (b.p.)

The four DNA nucleotides are adenine, guanine, cytosine, and thymine, commonly abbreviated to A, G, C, and T respectively. Each strand, according to chemical convention, has a 5' and a 3' end; thus, any single strand has a natural orientation (Figure 1). The classical double helix of DNA is formed when two separate strands bond. Bonding occurs by the pairwise attraction of bases; A bonds with T and G bonds with C. The pairs (A,T) and (G,C) are therefore known as *complementary* base pairs. The two pairs of bases form *hydrogen bonds* between each other, two bonds between A and T, and three between G and C

The bonding process, known as *annealing*, is fundamental to our implementation. A strand will only anneal to its complement if they have opposite polarities. Therefore, one strand of the double helix extends from 5' to 3', and the other from 3' to 5', as depicted in Figure 1.

II Operations on DNA

Some (but not all) DNA-based computations apply a specific sequence of biological operations to a set of strands. These operations are all commonly used by molecular biologists, and we now describe them in more detail.

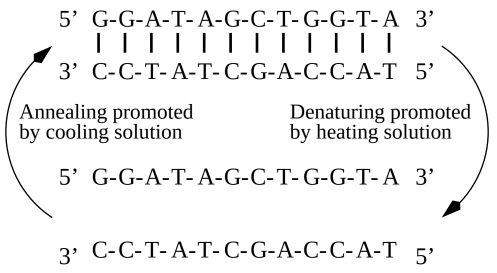
5’ G-G-A-T- A-G-C-T- G-G-T- A 3’
3’ C-C-T- A-T- C-G-A-C-C-A-T 5’
Annealing promoted by cooling solution
Denaturing promoted by heating solution
5’ G-G-A-T- A-G-C-T- G-G-T- A 3’
3’ C-C-T- A-T- C-G-A-C-C-A-T 5’

Figure 2: DNA melting and annealing (from [3]).

III Synthesis

Oligonucleotides may be synthesized to order by a machine the size of a microwave oven. The synthesizer is supplied with the four nucleotide bases in solution, which are combined according to a sequence entered by the user. The instrument makes millions of copies of the required oligo and places them in solution in a small vial.

IV Denaturing, annealing, and ligation

Double-stranded DNA may be dissolved into single strands (or *denatured*) by heating the solution to a temperature determined by the composition of the strand [11]. Heating breaks the hydrogen bonds between complementary strands (Figure 2). *Annealing* is the reverse of melting, whereby a solution of single strands is cooled, allowing complementary strands to bind together (Figure 2).

In double-stranded DNA, if one of the single strands contains a discontinuity (i.e., one nucleotide is not bonded to its neighbor) then this may be repaired by DNA *ligase* [12]. This particular enzyme is useful for DNA computing, as it allows us to create a unified strand from several strands bound together by their respective complements. Ligase therefore acts as a molecular “cement” or “mortar”, binding together several strands into a single strand.

V Separation of strands

Separation is a fundamental operation, and involves the extraction from a test tube of any *single* strands containing a specific short sequence (e.g., extract all strands containing the sequence $GCTA$). For this, we may use a “molecular sieving” process known as *affinity purification.* If we want to extract from a solution single strands containing the sequence x, we may first create many copies of its complement, $\overline{x}$. We attach to these oligos biotin molecules (a process known as “biotinylation”) which in turn bind to a fixed matrix. If we pour the contents of the test tube over this matrix, strands containing x will anneal to the anchored complementary strands. Washing the matrix removes all strands that *did not* anneal, leaving only strands containing x. These may then be removed from the matrix.

VI Gel electrophoresis

Gel electrophoresis is an important technique for sorting DNA strands by size [12]. Electrophoresis is the movement of charged molecules in an electric field. Since DNA molecules carry a negative charge, when placed in an electric field they tend to migrate toward the positive pole. The rate of migration of a molecule in an *aqueous* solution depends on its shape and electric charge. Since DNA molecules have the same charge per unit length, they all migrate at the same speed in an aqueous solution. However, if electrophoresis is carried out in a *gel* (usually made of agarose, polyacrylamide, or a combination of the two), the migration rate of a molecule is also affected by its *size*. This is due to the fact that the gel is a dense network of pores through which the molecules must travel. Smaller molecules therefore migrate faster through the gel, thus sorting them according to size. In order to sort strands, the DNA is placed in a well cut out of the gel, and a charge applied. After running the gel, the results are visualised by staining the DNA with dye and then viewing the gel under ultraviolet light. Bands of DNA of a specific length may then be cut out of the gel and soaked to free the strands (which may then be used again in subsequent processing steps).

VII PCR

The DNA *polymerases* perform several functions, including the repair and duplication of DNA. Given a short *primer* (or "tag") oligo, in the presence of nucleotide triphosphates (i.e., "spare" nucleotides), the polymerase extends the primer if and only if the primer is bound to a longer *template* strand.

Primer extension is fundamental to the *Polymerase Chain Reaction*, or PCR [29]. PCR is a process that quickly amplifies the amount of DNA in a given solution. PCR employs polymerase to make copies of a specific region (or *target sequence*) of DNA that lies between two *known* sequences. Note that this target sequence (which may be up to around 3,000 b.p. long) can be unknown ahead of time. In order to amplify template DNA with known regions (perhaps at either end of the strands), we first design forward and backward primers (i.e. primers that go from 5' to 3' on each strand. We then add a large excess (relative to the amount of DNA being replicated) of primer to the solution and heat it to denature the double-stranded template. Cooling the solution then allows the primers to anneal to their target sequences. We then add the polymerase, which extends the primers, forming an identical copy of the template DNA.

If we start with a single template, then of course we now have two copies. If we then repeat the cycle of heating, annealing, and polymerising, it is clear that this approach yields an exponential number of copies of the template (since the number of strands doubles after each cycle). A typical number of cycles would be perhaps 35, yielding (assuming a single template) around 68 billion copies of the *target sequence* (for example, a gene).

IV The First DNA Computation

We now describe in detail the first ever successful computation performed using strands of DNA. This experiment was carried out in 1994 by Leonard Adleman

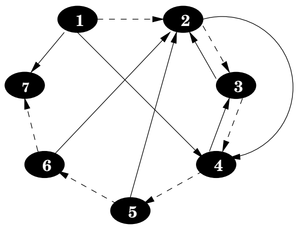

Figure 3: Instance of the HPP solved by Adleman (from [3]).

of the University of Southern California. Adleman was already a highly distinguished computer scientist prior to this work, having been awarded, along with Rivest and Shamir, the 2002 ACM Turing Award for the development of the RSA public-key encryption scheme [33].

Adleman utilized the incredible storage capacity of DNA to implement a brute-force algorithm for the directed Hamiltonian Path Problem (HPP) [1]. The HPP involves finding a path through a graph (or "network") that visits each vertex ("node") exactly once. For an example application, consider a salesperson who wishes to visit several cities connected by rail links. In order to save time and money, she would like to know if there exists an itinerary that visits every city precisely once. We model this situation by constructing a graph where vertices represent cities and edges the rail links. The HPP is a classic problem in the study of *complex networks and graph theory* (q.v.) [20], and belongs to the class of problems referred to as *NP-complete* [19]. To practitioners in the field of *computational complexity* (q.v.), the NP-complete problems are most interesting, since they are amongst the hardest known problems, and include many problems of great theoretical and practical significance, such as network design, scheduling, and data storage.

The instance of the HPP that Adleman solved is depicted in Figure 3, with the unique Hamiltonian Path (HP) highlighted by a dashed line. Adleman's approach was simple:

1. Generate strands encoding random paths such that the Hamiltonian Path (HP) is represented with high probability. The quantities of DNA used far exceeded those necessary for the small graph under consideration, so it is likely that *many* strands encoding the HP were present.

2. Remove all strands that do not encode the HP.

3. Check that the remaining strands encode a solution to the HPP.

The individual steps were implemented as follows:

Stage 1: Each vertex and edge was assigned a distinct 20-base sequence of DNA (Figure 4a). This implies that strands encoding a HP were of length 140 b.p., since there are seven vertices in the problem instance. Sequences representing edges act as 'splints' between strands representing their endpoints (Figure 4b).

In formal terms, the sequence associated with an edge $i \rightarrow j$ is the 3' 10-mer of the sequence representing v_i followed by the 5' 10-mer of the sequence

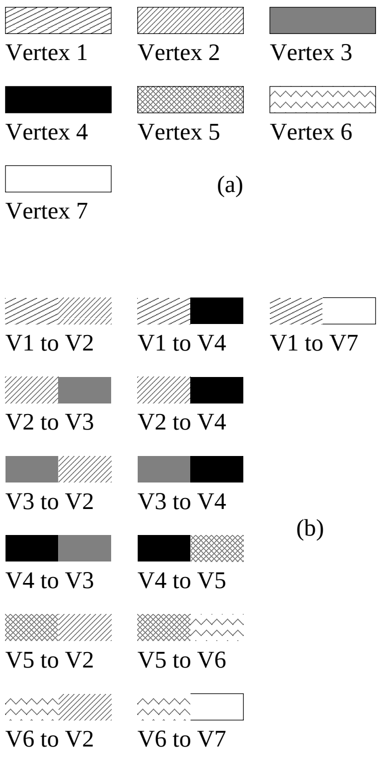

Figure 4: Adleman's scheme for encoding paths - schematic representation of oligos (from [3]).

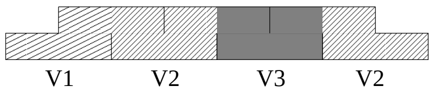

Figure 5: Example path created in Adleman's scheme (from [3]).

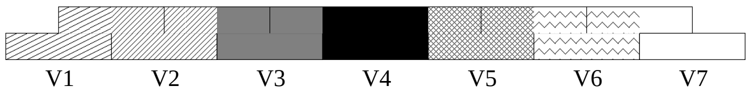

Figure 6: Unique Hamiltonian path (from [3]).

representing v_j. These oligos were then combined to form strands encoding random paths through the graph. An (illegal) example path ($v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4$) is depicted in Figure 5.

Fixed amounts (50 pmol) of each oligo were synthesised, mixed together and then ligase was added to seal any backbone nicks and form completely unified strands from the shorter "building blocks". At the end of this reaction, it is assumed that a complete strand representing the HP is present with high probability. This approach solves the problem of generating an exponential number of different paths using a polynomial number of initial oligos.

Stage 2: PCR was first used to massively amplify the population of strands encoding paths starting at v_1 and ending at v_7. These strands were amplified to such a massive extent that they effectively "overwhelmed" any strands that *did not* start and end at the correct points.

Next, strands that do not encode paths containing exactly n visits were removed. The product of the PCR amplification was run on an agarose gel to isolate strands of length 140 b.p. A series of affinity purification steps was then used to isolate strands encoding paths that visited each vertex exactly once.

Stage 3: PCR was used to identify the strand encoding the unique HP that this problem instance provides. For an n-vertex graph, we run $n-1$ PCR reactions, with the strand representing v_1 as the left primer and the complement of the strand representing v_i as the right primer in the i^{th} lane. The presence of molecules encoding the unique HP depicted in Figure 3 should produce bands of length 40, 60, 80, 100, 120, and 140 b.p. in lanes 1 through 6, respectively. This is exactly what Adleman observed.

V Models of DNA Computation

Although Adleman provided the impetus for subsequent work on DNA computing, his algorithm was not expressed within a formal model of computation. Richard Lipton realised that Adleman's approach, though seminal, was limited in that it was specific to solving the HPP. Lipton proposed extending Adleman's algorithm 'in a way that allows biological computers to potentially radically change the way we do all computations, not just HPPs.'[26]

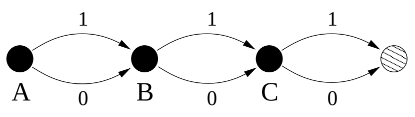

Figure 7: Lipton's graph representation for a three-bit binary string

I Satisfiability Model

Lipton's article described a methodology for solving the *satisfiability problem* (SAT) [16] using DNA. In terms of *computational complexity* (q.v.), SAT is the "benchmark" NP-complete problem, and may be phrased as follows: given a finite set $V = \{v_1, v_2, \ldots, v_n\}$ of logical variables, we define a *literal* to be a variable, v_i, or its complement, $\overline{v_i}$. If v_i is $true$ then $\overline{v_i}$ is $false$, and vice-versa. We define a *clause*, C_j, to be a set of literals $\{v_1^j, v_2^j, \ldots, v_l^j\}$. An instance, I, of SAT consists of a set of clauses. The problem is to assign a Boolean value to each variable in V such that at least one variable in each clause has the value $true$. If this is the case we may say that I has been $satisfied$.

By showing how molecular computers could be applied to this archetypal NP-complete problem, Lipton hoped to show how *any* difficult problem may be solved using this approach. The underlying principle was the same as Adleman's approach: generate all possible solutions to the problem, then gradually *filter out* strands until any remaining strands *must* be a solution to the problem. Lipton proposed solving an instance of SAT by starting with a tube containing strands representing all possible assignments to its variables.

Lipton's main contribution lay in his method for encoding *arbitrary* binary strings as strands of DNA. If we only need a small number different strands as the foundation of our computer, then it would be fairly cheap and quick to specify and order them to be individually synthesised. However, Lipton realized that this approach would quickly become infeasible as the number of variables grew. One of the characteristics of the NP-complete problems is that for even a small increase in the problem size (in this case, the number of variables), the number of possible solutions rises exponentially. For any non-trivial problem, Lipton would require a prohibitively expensive number of 'one-off' strands. For example, if Lipton wanted to solve a problem with ten variables, he would need to order $2^{10} = 1,024$ individual strands.

Lipton needed a way of encoding an *exponential*-sized pool of starting strands, using only a polynomial number of 'building-block' strands. Just as Adleman encoded a large number of possible paths through a graph, using a small number of node and edge strands, Lipton wanted to build a large number of assignment strands, using a small number of 'variable' strands. His key insight was that an arbitrary binary string (in this case encoding the values of a set of variables) could be represented as a path through a graph, where each node represented one particular bit (or variable).

Figure 7 shows an example of a graph to encode a string of three bits (variables), each represented by a node labelled A, B or C. Each node has two edges emanating from it, labelled either 1 or 0 (the edges leading out of node C lead to a 'dummy' node, which just acts like a railway buffer to end the paths). Any three-bit string can be represented by a path built by taking one of the two possible paths at each node, the value of each bit being defined by the label of the edge that is taken at each step. For example, if we only ever take the 'top' path at each branch, we encode the string 111, and if we only ever take the 'bottom' path, we end up with 000. A path that goes 'top, bottom, top' encodes the string 101, 'top, top, bottom' encodes 110, and so on.

The power of this encoding is that, just like Adleman's approach, we only have to generate one sequence for each node (including the dummy node) and one for each edge . With the correct Watson-Crick encoding, random paths through the graph form spontaneously, just like in Adleman's Hamiltonian Path experiment. Using this approach, Lipton believed he could be confident of starting off with a tube containing all possible binary strings of a given length. Each strand in the tube would encode a possible assignment of values for a set of variables. The next stage was to remove strands encoding those assignments that did *not* result in satisfaction of the formula to be solved.

Because the general form of a SAT formula is a set of clauses combined with the AND operation, it follows that each clause *must* be satisfied: if evaluating only a single clause results in a value of 0, then the *whole* formula evaluates to 0, and the formula is not satisfied. Because the variables in each clause are separated by the OR operation, it only takes a single variable to take the value 1 for the whole clause to be satisfied. Lipton's algorithm was very straightforward: he would proceed one clause at a time, looking at each component of it in turn, and keeping only the strands encoding a sequence of bits that satisfied that particular clause. The 'winning' strands would then go forward to the *next* clause, where the process would repeat, until there were no more clauses to examine. At the end of this procedure, if Lipton had *any* strands left in his tube, then he would know with certainty that the formula was satisfiable, since only strings satisfying every clause would have survived through the successive filters.

We now show how Lipton's method can be *formally* expressed. In all *filtering* models of DNA computing, a computation consists of a sequence of operations on finite *multi-sets* of strings. Multi-sets are sets that may contain more than one copy of the same element. It is normally the case that a computation begins and terminates with a single multi-set. Within the computation, by applying legal operations of a model, several multi-sets may exist at the same time. We define operations on multi-sets shortly, but first consider the nature of an *initial set.*

An initial multi-set consists of strings which are typically of length $O(n)$ where n is the problem size. As a subset, the initial multi-set should include all possible solutions (each encoded by a string) to the problem to be solved. The point here is that the superset, in any implementation of the model, is supposed to be relatively easy to generate as a starting point for a computation. The computation then proceeds by *filtering out* strings which cannot be a solution.

Within one possible model [2], the following operations are available on sets of strings over some alphabet α:

- *separate*(T, S). Given a set T and a substring S, create two new sets $+(T, S)$ and $-(T, S)$, where $+(T, S)$ is all strings in T containing S, and $-(T, S)$ is all strings in T *not* containing S.

- *merge*$(T_1, T_2, \ldots, T_n)$. Given set $T_1, T_2, \ldots, T_n$, create $\cup(T_1, T_2, \ldots, T_n) = T_1 \cup T_2 \cup \ldots T_n$.

- *detect*(T). Given a set T, return *true* if T is nonempty, otherwise return $false$.

For example, given $\alpha = \{A, B, C\}$, the following algorithm returns *true* only if the initial multi-set contains a string composed entirely of "A"s:

```
Input(T)
$T \leftarrow -(T, B)$
$T \leftarrow -(T, C)$
Output(detect($T$))
```

Although Lipton does not explicitly define his operation set in [26], his solution to SAT may be phrased in terms of the the operations above, described by Adleman in [2]. Lipton employs the *merge*, *separate*, and *detect* operations described above. The initial set T contains many strings, each encoding a single n-bit sequence. All possible n-bit sequences are represented in T. The algorithm proceeds as follows:

```
(1) Create initial set, $T$
(2) For each clause do begin
(3)     For each literal $v_i$ do begin
(4)         if $v_i = x_j$ extract from $T$ strings encoding $v_i = 1$ else
            extract from $T$ strings encoding $v_i = 0$
(5)     End for
(6)     Create new set $T$ by merging extracted strings
(7) End for
(8) If $T$ nonempty then $I$ is satisfiable
```

The pseudo-code algorithm may be expressed more formally thus:

```
(1) Input(T)
(2) for $a = 1$ to $|I|$ do begin
(3)     for $b = 1$ to $|C_a|$ do begin
(4)         if $v_b^a = x_j$ then $T_b \leftarrow +(T, v_b^a = 1)$
            else $T_b \leftarrow +(T, v_b^a = 0)$
(5)     end for
(6)     $T \leftarrow merge(T_1, T_2, \ldots, T_b)$
(7) end for
(8) Output(detect($T$))
```

Step 1 generates all possible n-bit strings. Then, for each clause $C_a = \{v_1^a, v_2^a, \ldots, v_l^a\}$ (Step 2) we perform the following steps. For each literal v_b^a (Step 3) we operate as follows: if v_b^a computes the positive form then we extract from T all strings encoding 1 at position v_b^a, placing these strings in T_b; if v_b^a computes the negative form we extract from T all strings encoding 0 at position v_b^a, placing these strings in T_b (Step 4); after l iterations, we have satisfied every variable in clause C_a; we then create a new set T from the union of sets $T_1, T_2, \ldots, T_b$ (Step 6) and repeat these steps for clause $C_a + 1$ (Step 7). If any strings remain in T after all clauses have been operated upon, then I is satisfiable (Step 8). Although Lipton did not report an experimental verification of his algorithm, the biological implementation would be straightforward, using affinity purification to implement extraction, pouring of tubes for *merge* and running the solution through a gel to *detect*.

Soon after Lipton's paper, the Proceedings of the Second Annual Workshop on DNA Based Computers contained several papers describing significant developments in molecular computing. These included the parallel filtering model, the sticker model, DNA self-assembly, RNA computing and surface-based computing.

We describe the last four developments (or variants thereof) in the next section. Here, we briefly introduce the parallel filtering model, as it motivates some of the later discussion.

II Parallel Filtering Model

The first description of the parallel filtering model appeared in [4], with further analysis in [3]. This model was the first to provide a formal framework for the easy description of DNA algorithms for *any* problem in the complexity class NP. The main difference between the parallel filtering model and those previously proposed lies in the *implementation* of the removal of strings. All other models propose *separation* steps, where strings are *conserved*, and may be used later in the computation. Within the parallel filtering model, however, strings that are removed are *discarded*, and play no further part in the computation. This model is the first exemplar of the so-called "mark and destroy" paradigm of molecular computing, and was motivated in part by discussion of the error-prone nature of biotin-based separation techniques. The new operation introduced within this model is:

- $remove(U, \{S_i\})$. This operation removes from the set U, in parallel, any string which contains at least one occurrence of any of the substrings S_i.

The proposed implementation of the *remove* operation is as follows: for a given substring S_i, add "tag" strands composed of its Watson-Crick *complement* to the tube U, so that the tags anneal only to strands containing the target subsequence. We then add polymerase to make tagged strands double-stranded from the point of annealing ("mark"). We assume that working strands contain a specific restriction site embedded along them at fixed intervals, so adding the appropriate restriction enzyme removes only marked strands by repeated digestion ("destroy").

VI Subsequent Work

In this section we describe several other successful laboratory implementations of molecular-based computing, each illustrating a novel approach or new technique. Our objective is not to give an exhaustive description of each experiment, but to give a high-level treatment of the general methodology, so that the reader may approach with confidence the fuller description in the literature.

I DNA Addition

One of the first successful experiments reported after Adleman's result was due to Guarnieri *et al.* in 1996 [21], in which they describe a DNA-based algorithm

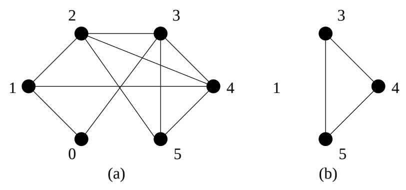

Figure 8: (a) Ouyang graph. (b) Example subgraph (from [3]).

for binary addition. The method uses single-stranded DNA reactions to add together two nonnegative binary numbers. This application, as the authors note, is very different from previous proposals, which use DNA as the substrate for a massively parallel random search.

Adding binary numbers requires keeping track of the position of each digit and of any "carries" that arise from adding 1 to 1 (remembering that 1 + 1 = 0 plus carry 1 in binary). The DNA sequences used represent not only binary strings but also allow for carries and the extension of DNA strands to represent answers. Guarnieri *et al.* use sequences that encode a digit in a given position and its significance, or position from the right. For example, the first digit in the first position is represented by two DNA strands, each consisting of a short sequence representing a "position transfer operator", a short sequence representing the digit's value, and a short sequence representing a "position operator."

DNA representations of all possible two bit binary integers are constructed, which can then be added in pairs. Adding a pair involves adding appropriate complementary strands, which then link up and provide the basis for strand extension to make new, longer strands. This is termed a "horizontal chain reaction", where input sequences serve as templates for constructing an extended result strand. The final strand serves as a record of successive operations, which is then read out to yield the answer digits in the correct order.

The results obtained confirmed the correct addition of 0 + 0, 0 + 1, 1 + 0, and 1 + 1, each calculation taking between 1 and 2 days of bench work. Although limited in scope, this experiment was (at the time) one of the few experimental implementations to support theoretical results.

II Maximal Clique Computation

The problem of finding a Maximal Clique using DNA was addressed by Ouyang *et al.* in 1997 [31]. A *clique* is a fully connected subgraph of a given graph (Figure 8). The *maximal clique* problem asks: given a graph, how many vertices are there in the largest clique? Finding the size of the largest clique is an NP-complete problem.

The algorithm proceeds as follows: for a graph G with n vertices, all possible vertex subsets (subgraphs) are represented by an n-bit binary string $b_{n-1}, b_{n-2}, \ldots, b_0$. For example, given the six-vertex graph used in [31] and depicted in Fig. 8a, the string 111000 corresponds to the subgraph depicted in Fig. 8b, containing v_5, v_4, and v_3.
Clearly, the largest clique in this graph contains v_5, v_4, v_3, and v_2, represented by the string 111100.

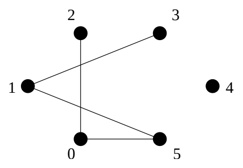

Figure 9: Complement of Ouyang graph (from [3]).

The next stage is to find pairs of vertices that are not connected by an edge (and, therefore, by definition, cannot appear together in a clique). We begin by taking the *complement* of G, $\overline{G}$, which contains the same vertex set as G, but which only contains an edge $\{v, w\}$ if $\{v, w\}$ is *not* present in the edge set of G. The complement of the graph depicted in Fig. 8 is shown in Fig. 9.

If two vertices in $\overline{G}$ are connected by an edge then their corresponding bits cannot both be set to 1 in any given string. For the given problem, we must therefore remove strings encoding ***1*1 (v_2 and v_0), 1****1 (v_5 and v_0), 1***1* (v_5 and v_1) and **1*1* (v_3 and v_1), where * means either 1 or 0. All other strings encode a (not necessarily maximal) clique.

We must then sort the remaining strings to find the largest clique. This is simply a case of finding the string containing the largest number of 1s, as each 1 corresponds to a vertex in the clique. The string containing the largest number of 1s encodes the largest clique in the graph.

The DNA implementation of the algorithm goes as follows. The first task is to construct a set of DNA strands to represent all possible subgraphs. There are two strands per bit, to represent either 1 or 0. Each strand associated with a bit i is represented by two sections, its position P_i and its value V_i. All P_i sections are of length 20 bases. If $V_i = 1$ then the sequence representing V_i is a restriction site unique to that strand. If $V_i = 0$ then the sequence representing V_i is a 10 base "filler" sequence. Therefore, the longest possible sequence is 200 bases, corresponding to the string 000000, and the shortest sequence is 140 bases, corresponding to 111111.

The computation then proceeds by digesting strands in the library, guided by the complementary graph $\overline{G}$. To remove strands encoding a connection i, j in $\overline{G}$, the current tube is divided into two, t_0 and t_1. In t_0 we cut strings encoding $V_i = 1$ by adding the restriction enzyme associated with V_i. In t_1 we cut strings encoding $V_j = 1$ by adding the restriction enzyme associated with V_j. For example, to remove strands encoding the connection between V_0 and V_2, we cut strings containing $V_0 = 1$ in t_0 with the enzyme Afl II, and we cut strings containing $V_2 = 1$ in t_1 with Spe I. The two tubes are then combined into a new working tube, and the next edge in $\overline{G}$ is dealt with.

In order to read the size of the largest clique, the final tube was simply run on a gel. The authors performed this operation, and found the shortest band to be 160 bp, corresponding to a 4-vertex clique. This DNA was then sequenced and found to represent the correct solution, 111100.

Although this is another good illustration of a DNA-based computation, the authors acknowledge the lack of scalability of their approach. One major factor is the requirement that each vertex be associated with an individual restriction enzyme. This, of course, limits the number of vertices that can be handled by the number of restriction enzymes available. However, a more fundamental issue is the exponential growth in the problem size (and thus the initial library), which we shall encounter again.

III Chess Games

In [17], Faulhammer *et al.* describe a solution to a variant of the satisfiability problem that uses RNA rather than DNA as the computational substrate. They consider a variant of SAT, the so-called "Knight problem", which seeks configurations of knights on an $n \times n$ chess board, such that no knight is attacking any other [41].

The authors prefer the "mark and destroy" strategy rather than the repeated use of extraction to remove illegal solutions. However, the use of an RNA library and ribonuclease (RNase) H digestion gives greater flexibility, as one is not constrained by the set of restriction enzymes available. In this way, the RNase H acts as a "universal restriction enzyme", allowing selective marking of virtually any RNA strands for parallel destruction by digestion.

The particular instance solved in [17] used a 3×3 board, with the variables $a - i$ representing the squares. If a variable is set to 1 then a knight is present at that variable's square, and 0 represents the absence of a knight. The 3×3 knight problem may therefore be represented as the following instance of SAT:

$$((\neg h \wedge \neg f) \vee \neg a) \wedge ((\neg g \wedge \neg i) \vee \neg b) \wedge ((\neg d \wedge \neg h) \vee \neg c) \wedge ((\neg c \wedge \neg i) \vee \neg d) \wedge ((\neg a \wedge \neg g) \vee \neg f) \wedge ((\neg b \vee \neg f) \vee \neg g) \wedge ((\neg a \wedge \neg c) \vee \neg h) \wedge ((\neg d \wedge \neg b) \vee \neg i)$$

which, in this case, simplifies to

$$((\neg h \wedge \neg f) \wedge \neg a) \wedge ((\neg g \wedge \neg i) \vee \neg b) \wedge ((\neg d \wedge \neg h) \vee \neg c) \wedge ((\neg c \wedge \neg i) \vee \neg d) \wedge ((\neg a \wedge \neg g) \vee \neg f).$$

This simplification greatly reduces the number of laboratory steps required. The experiment proceeds by using a series of RNase H digestions of "illegal" board representations, along the lines of the parallel filtering model [4].

Board representations are encoded as follows: the experiment starts with all strings of the form $x_1, \ldots, x_n$, where each variable x_i takes the value 1 or 0; then, the following operations may be performed on the population of strings:

- Cut all strings containing any pattern of specified variables $p_i, \ldots, p_k$
- Separate the "test tube" into several collections of strings (molecules) by length
- Equally divide (i.e., split) the contents of a tube into two tubes
- Pour (mix) two test tubes together
- Sample a random string from the test tube

The first stage of the algorithm is the construction of the initial library of strands. Each strand sequence follows the template depicted in Fig. 10.

The prefix and suffix regions are included to facilitate PCR. Each variable is represented by one of two unique sequences of length 15 nucleotides, one

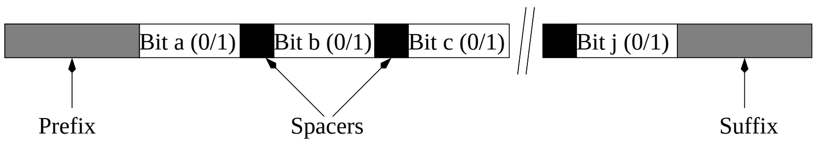

Figure 10: Template for RNA strands (from [3]).

representing the fact that the variable is set to 1, and the other the fact that it is set to 0. Variable regions are separated by short (5 nucleotide) spacer regions. In order to avoid having to individually generate each individual sequence, a "mix and split" strategy (described in more detail in [17]) is used. The RNA version of the library is then generated by *in vitro* transcription.
The algorithm proceeds as follows:

1. For each square, sequentially, split the RNA library into two tubes, labelled 1 and 2. After digestions have taken place, tube 1 will contain strands that contain a knight at that square, and tube 2 will contain strands that do *not* have knights at that square

2. In tube 1, digest with RNase H strands that have no knight at position **a**, as well as strands that describe a knight at attacking positions **h** and **f**. This implements the logical statement $((\neg h \wedge \neg f) \vee \neg a)$

3. In tube 2, digest strands that have a knight present at position **a**

4. Remove the DNA oligos used to perform the above digestions

5. Go to step 1, repeating with square **b**

Steps 1 through 4 implement the following: "There may or may not be a knight in square **a**: if there *is*, then it is attacking squares **h** and **f**, so disallow this." The algorithm only needs to be performed for squares **a, b, c, d,** and **f**, as square **e**, by the rules of chess, cannot threaten or be threatened on a board this size, and any illegal interactions that squares **g, h,** and **i** may have are with **a, b, c, d,** and **f**, and have already been dealt with. At the conclusion of this stage, any remaining full-length strands are recovered, as they should encode legal boards.

The "mark and destroy" digestion operation is implemented as follows. If we wish to retain (i.e., select) strands encoding variable a to have value 1, DNA oligonucleotides corresponding to the complement of the $a = 0$ sequence are added to the tube, and anneal to all strands encoding $a = 0$. RNase H is then added to the solution. Ribonuclease H (RNase H) is an endoribonuclease which specifically hydrolyzes the phosphodiester bonds of RNA hybridized to DNA. RNase H does not digest single or double-stranded DNA, so his operation therefore leaves intact only those strands encoding $a = 1$, in a fashion similar to the removal operation of the parallel filtering model [4].

The results obtained (described in [17]) were extremely encouraging: out of 43 output strands sampled, only one contained an illegal board. Given that the population sampled encoded 127 knights, this gave an overall knight placement success rate of 97.7%.

Table 1: Strands used to represent SAT variable values

Strand	*Sequence*	*wxyz*
S_0	CAACCCAA	0000
S_1	TCTCAGAG	0001
S_2	GAAGGCAT	0010
S_3	AGGAATGC	0011
S_4	ATCGAGCT	0100
S_5	TTGGACCA	0101
S_6	ACCATTGG	0110
S_7	GTTGGGTT	0111
S_8	CCAAGTTG	1000
S_9	CAGTTGAC	1001
S_10	TGGTTTGG	1010
S_11	GATCCGAT	1011
S_12	ATATCGCG	1100
S_13	GGTTCAAC	1101
S_14	AACCTGGT	1110
S_15	ACTGGTCA	1111

IV Computing on Surfaces

Another experiment that makes use of the "mark and destroy" paradigm is described in [27] (although early work was performed from 1996). The key difference between this and previous experiments is that the DNA strands used are tethered to a support rather than being allowed to float freely in solution. The authors argue that this approach greatly simplifies the automation of the (potentially very many) repetitive chemical processes required during the performance of an experiment.

The authors report a DNA-based solution to a small instance of SAT. The specific problem solved is

$$(w \vee x \vee y) \wedge (w \vee \neg y \vee z) \wedge (\neg x \vee y) \wedge (\neg w \vee \neg y).$$

16 unique DNA strands were synthesized, each one corresponding to one of the $2^4 = 16$ combinations of variable values. The actual encodings are given in Table 1 (taken from [27]).
Each of the 16 sets of strands was then affixed to a specific region of a gold coated surface, so that each solution to the SAT problem was represented as an individual cluster of strands.

The algorithm then proceeds as follows. For each clause of the problem, a cycle of "mark", "destroy", and "unmark" operations is carried out. The goal of each cycle is to destroy the strands that do *not* satisfy the appropriate clause. Thus, in the first cycle, the objective is to destroy strands that do not satisfy the clause $(w \vee x \vee y)$. Destruction is achieved by "protecting", or *marking* strands that *do* satisfy the clause by annealing to them their complementary strands. *E.coli* exonuclease I is then used to digest unmarked strands (i.e., any single-stranded DNA).

By inspection of Table 1, we see that this applies to only two strands, S_0

(0000) and S_1 (0001). Thus, in cycle 1 the complements of the 14 other strands

($w = 1(S_8, S_9, S_10, S_11, S_12, S_13, S_14, S_15)$;
$x = 1(S_4, S_5, S_6, S_7, S_12, S_13, S_14, S_15)$;
$y = 1(S_2, S_3, S_6, S_7, S_10, S_11, S_14, S_15)$)

were combined and hybridized to the surface before the exonuclease I was added. The surface was then regenerated (the unmark operation) to return the remaining surface-bound oligos (S_2-S_15) to single-stranded form. This process was repeated three more times for the remaining three clauses, leaving a surface containing only strands encoding a legal solution to the SAT problem.

The remaining molecules were amplified using PCR and then hybridized to an addressed array. The results of fluorescence imaging clearly showed four spots of relatively high intensity, corresponding to the four regions occupied by legal solutions to the problem (S_3, S_7, S_8, and S_9).

Although these results are encouraging as a first move toward more error-resistant DNA computing, as the authors themselves acknowledge, there remain serious concerns about the scalability of this approach (1,536 individual oligos would be required for a 36-bit, as opposed to 4-bit, implementation).

V Computing with Hairpins

The tendency of DNA molecules to self-anneal was exploited by Sakamoto *et al.* in [36] for the purposes of solving a small instance of SAT. The authors encode the given formula in "literal strings" which are conjunctions of the literals selected from each SAT clause (one literal per clause). A formula is satisfiable if there exists a literal string that does not contain any variable together with its negation. If each variable is encoded as a DNA subsequence that is the Watson-Crick complement of its negation then any strands containing a variable and its negation self-anneal to form "hairpin" structures. These can be distinguished from non-hairpin structure-forming strands, and removed. The benefit of this approach is that it does not require physical manipulation of the DNA, only temperature cycling. The drawback is that it requires 3^m literal strings for m clauses, thus invoking once again the scalability argument.

VI Gel-Based Computing

A much larger (20 variable) instance of 3-SAT was successfully solved by Adleman's group in an experiment described in 2002 [9]. This is, to date, the largest known problem instance successfully solved by a DNA-based computer; indeed, as the authors state, "this computational problem may yet be the largest yet solved by nonelectronic means" [9].

The architecture underlying the experiment is related to the Sticker Model described by Roweis *et al.* [35]. The difference here is that only separation steps are used – the application of stickers is not used. Separations are achieved by using oligo probes immobilized in polyacrylamide gel-filled glass modules, and strands are pulled through them by electrophoresis. Strands are removed (i.e., retained in the module) by virtue of their hybridizing to the immobilized probes, with other strands free to pass through the module and be subject to further processing. Captured strands may be released and transported (again via electrophoresis) to other modules for further processing.

The potential benefits of such an approach are clear; the use of electrophoresis minimizes the number of laboratory operations performed on strands, which, in turn, increases the chance of success of an experiment. Since strands are not deliberately damaged in any way, they, together with the glass modules, are potentially reusable for multiple computations. Finally, the whole process is potentially automatable.

The problem solves was a 20-variable, 24-clause 3-SAT formula Φ, with a unique satisfying truth assignment. These are

$$\Phi = (\neg x_{13} \vee x_{16} \vee x_{18}) \wedge (x_5 \vee x_{12} \vee \neg x_9) \wedge (\neg x_{13} \vee \neg x_2 \vee x_{20}) \wedge (x_{12} \vee x_9 \vee \neg x_5) \wedge (x_{19} \vee \neg x_4 \vee x_6) \wedge (x_9 \vee x_{12} \vee \neg x_5) \wedge (\neg x_1 \vee x_4 \vee \neg x_{11}) \wedge (x_{13} \vee \neg x_2 \vee \neg x_{19}) \wedge (x_5 \vee x_{17} \vee x_9) \wedge (x_{15} \vee x_9 \vee \neg x_{17}) \wedge (\neg x_5 \vee \neg x_9 \vee \neg x_{12}) \wedge (x_6 \vee x_{11} \vee x_4) \wedge (\neg x_{15} \vee \neg x_{17} \vee x_7) \wedge (\neg x_6 \vee x_{19} \vee x_{13}) \wedge (\neg x_{12} \vee \neg x_9 \vee x_5) \wedge (x_{12} \vee x_1 \vee x_{14}) \wedge (x_{20} \vee x_3 \vee x_2) \wedge (x_{10} \vee \neg x_7 \vee \neg x_8) \wedge (\neg x_5 \vee x_9 \vee \neg x_{12}) \wedge (x_{18} \vee \neg x_{20} \vee x_3) \wedge (\neg x_{10} \vee \neg x_{18} \vee \neg x_{16}) \wedge (x_1 \vee \neg x_{11} \vee \neg x_{14}) \wedge (x_8 \vee \neg x_7 \vee \neg x_{15}) \wedge (\neg x_8 \vee x_{16} \vee \neg x_{10})$$

with a unique satisfying assignment of

$$x_1 = F, x_2 = T, x_3 = F, x_4 = F, x_5 = F, x_6 = F, x_7 = T, x_8 = T, x_9 = F, x_{10} = T, x_{11} = T, x_{12} = T, x_{13} = F, x_{14} = F, x_{15} = T, x_{16} = T, x_{17} = T, x_{18} = F, x_{19} = F, x_{20} = F.$$

As there are 20 variables, there are $2^{20} = 1,048,576$ possible truth assignments. To represent all possible assignments, two distinct 15 base *value sequences* were assigned to each variable $x_k (k = 1, \ldots, 20)$, one representing true (T), X_k^T, and one representing false (F), X_k^F. A mix and split generation technique similar to that of Faulhammer *et al.* [17] was used to generate a 300-base *library sequence* for each of the unique truth assignments. Each library sequence was made up of 20 value sequences joined together, representing the 20 different variables. These library sequences were then amplified with PCR.

The computation proceeds as follows: for each clause, a glass clause module is constructed which is filled with gel and contains covalently bound probes designed to capture only those library strands that *do satisfy* that clause; strands that do not satisfy the clause are discarded.

In the first clause module $(\neg x_{13} \vee x_{16} \vee x_{18})$ strands encoding X_3^F, X_{16}^F, and X_{18}^T are retained, while strands encoding X_3^T, X_{16}^T, and X_{18}^F are discarded. Retained strands are then used as input to the next clause module, for each of the remaining clauses. The final (24^{th}) clause module should contain only those strands that have been retained in all 24 clause modules and hence encode truth assignments satisfying Φ.

The experimental results confirmed that a unique satisfying truth assignment for Φ was indeed found using this method. Impressive though it is, the authors still regard with scepticism claims made for the potential superiority of DNA-based computers over their traditional silicon counterparts. In a separate interview, Len Adleman stated that “DNA computers are unlikely to become stand-alone competitors for electronic computers. We simply cannot, at this time, control molecules with the deftness that electrical engineers and physicists control electrons” [32]. However, “they [DNA computers] enlighten us about alternatives to electronic computers and studying them may ultimately lead us to the true 'computer of the future'" [9]. In the next Section, we consider how the focus of DNA computing has since shifted away from the solution of "traditional" problems.

VII Assessment

In February 1995, one of the founding fathers of the theory of *computational complexity* (q.v.) published a short paper on DNA computing. In his article, titled "On the Weight of Computations" [22], Juris Hartmanis sounded a cautionary note amidst the growing interest surrounding DNA computing. By calculating the smallest amount of DNA required to encode *all possible paths* through a graph, he calculated that Adleman's experiment, if scaled up from 7 cities to 200 cities, would require an initial set of DNA strands that would weigh more than the Earth. Hartmanis was quick to point out the value of the initial work – "Adleman's molecular solution of the Hamiltonian path problem is indeed a magnificent achievement and may inititiate a more intensive exploration of molecular computing and computing in biological systems." However, he also emphasised the long-term futility of hoping that DNA-based computers could ever beat silicon machines in this particular domain. Soon after Adleman's experiment, hopes were expressed that a lot of the promise of molecular computers could be derived from their massive inherent parallelism – each operation is performed at the same time on trillions of DNA strands. However, as Turing showed, computers are not limited to any particular physical construction, and a DNA computer will suffer just as much as its silicon counterpart from the "exponential curse". If we require a molecular algorithm for a difficult problem to run in reasonable *time* then there is a fundamental requirement (unless P is shown to be equivalent to NP) for an exponential amount of *space* (in this case, the amount of DNA required). As Hartmanis observed, "...the exponential function grows too fast and the atoms are a bit too heavy to hope that the molecular computer can break the exponential barrier, this time the weight barrier."

This view would now seem to largely reflect the community consensus. As Ogihara and Ray argued in 2000, "It is foolish to attempt to predict the future of technology, but it may be that the ideal application for DNA computation does not like in computing large NP problems" [30]. In an interview in 2002, nanotechnologist Ned Seeman stated that "I am not a computer scientist, but I suspect it [molecular computing] is not well suited to traditional problems. I certainly feel that it is better suited to algorithmic self-assembly and biologically-oriented applications" [37]. That is not to say that molecular-based computations do not have a future, and in the final Section we briefly discuss some possible future directions that molecular computing may take (and offer pointers to other articles in the current volume).

VIII Future Directions

If molecular computations are to be scalable and/or sustainable then it is clear that they should be performed with a minimum of human intervention. The type of experiment originally performed by Adleman was feasible because it required a relatively small number of biological steps (although it still took a week of bench time to carry out). However, the number of manipulations (and the amount of material) required for a non-trivial computation would quickly render this approach infeasible. Although attempts have been made to automate molecular computations using, for example, microfluidics [40], this does not allow us to avoid the fundamental issue of scalability.

One promising subfield concerns molecular computing using machines (or "automata") constructed from DNA strands and other biomaterials. The construction of *molecular automata* (q.v.) was demonstrated by Benenson *et al.* in [6]. This experiment builds on the authors' earlier work [7] on the construction of biomolecular machines. In [6], the authors describe the construction of a molecular automaton that uses the process of DNA backbone hydrolysis and strand hybridization, fuelled by the potential free energy stored in the DNA itself.

One aim of researchers in this field is to build automata that may operate within living cells. Such devices may offer possibilities in terms of novel therapeutics, drug synthesis, bio-nanotechnology or *amorphous computing* (q.v.) Theoretical studies in DNA computing based on abstract models of the cell are already well-established (see the article on *membrane computing*), but recent work on *bacterial computing* (q.v.) has illustrated the feasibility of engineering living cells for the purposes of human-defined computation.

Finally, the notion of *biomolecular self-assembly* (q.v.) suggests ways in which the tendency of molecules to form spontaneously ordered structures may be harnessed, either for the purposes of computation or for engineering-based applications (for example, *DNA-templated self-assembly of proteins and nanowires* (q.v.). *Algorithmic* self-assembly has been demonstrated in the laboratory by Mao *et al.* [28]. This builds on work done on the self-assembly of periodic two-dimensional arrays (or "sheets") of DNA tiles connected by "sticky" pads [45, 46]. The authors of [28] report a one-dimensional algorithmic self-assembly of DNA triple-crossover molecules (tiles) to execute four steps of a logical XOR operation on a string of binary bits.

Triple-crossover molecules contain four strands that self-assemble through Watson-Crick complementarity to produce three double helices in roughly a planar formation. Each double helix is connected to adjacent double helices at points where their strands cross over between them. The ends of the core helix are closed by hairpin loops, but the other helices may end in sticky ends which direct the assembly of the macrostructure. The tiles then self-assemble to perform a computation. The authors of [28] report successful XOR computations on pairs of bits, but note that the scalability of the approach relies on proper hairpin formation in very long single-stranded molecules, which cannot be assumed.

In 2006, Paul Rothemund produced a ground-breaking piece of work, which he called "DNA origami" [34]. His breakthrough was to describe a competely novel method for constructing *arbitrary* two-dimensional DNA-based structures. "When it comes to making shapes out of DNA," explained Lloyd Smith in a commentary article[38], "the material is there, and its properties are understood. What was missing was a convincing, universal design scheme to allow our capabilities to unfold to the full." Rothemund demonstrated the generality of his approach by showing how it could build structures as diverse as a triangle, a five-pointed star, and even a “smiley” face and a map of the Americas. The shapes that he managed to construct were ten times more complex than anything that had been constructed from DNA before. Rothemund’s scheme was different to previous tile-based methods in that it used only a *single* long DNA strand at its foundation. By repeatedly folding a long strand of DNA around in a maze-like pattern, a scaffold was formed that traced the outline of the desired object. This structure was then “clipped” into place by short DNA strands, which Rothemund referred to as “staples”. He used as the main building block a 7,000 base sequence of DNA taken from the M13 virus. He developed a computer program that would take a human-designed folding path (the “shape”), map it onto the sequence of the DNA strand and then generate the sequences of the staples that would be required to hold it all together. In order to demonstrate the power of his method, he built a DNA map of the Americas, where a single nanometer (one billionth of a meter) represented 200 kilometres in real life (the map was 75 nanometres across). “The results that emerge are stunning,” praised Lloyd Smith, after seeing the work. The possible impact of this work may well be huge, with one possible application being a “nanoworkbench”. In 2003, John Reif and his team had shown how a simple DNA scaffold could cause proteins to self-assemble into regular, periodic two-dimensional grids [47]. By using his approach to build much more complex holding structures, Rothemund argued that it could now be used to carry out biological experiments at the nanoscale, studying the behaviour of complex protein assemblies, such as drugs, in a spatially-controlled environment. As Lloyd Smith observes, “Thus equipped not only with DNA building materials and an understanding of their structural and chemical properties, but also with a versatile general approach to weaving them together, we are arriving at a new frontier in our pursuit of ever-smaller structures. The barrier we have to surmount next is to deploy our knowledge to develop structures and devices that are really useful. Happily, in that endeavour we are now perhaps limited more by our imagination than our ability”[38].

Primary Literature

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

[2] Leonard M. Adleman. On constructing a molecular computer. Draft, University of Southern California, 1995.

[3] Martyn Amos. *Theoretical and Experimental DNA Computation*. Springer, 2005.

[4] Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant implementation of DNA computations. In Landweber and Baum [25].

[5] A. Arkin and J. Ross. Computational functions in biochemical reaction networks. *Biophysical Journal*, 67:560–578, 1994.

[6] Yaakov Benenson, Rivka Adam, Tamar Paz-Livneh, and Ehud Shapiro. DNA molecule provides a computing machine with both data and fuel. *Proceedings of the National Academies of Science*, 100(5):2191–2196, 2003.

[7] Yaakov Benenson, Tamar Paz-Elizur, Rivka Adar, Ehud Keinan, Zvi Livneh, and Ehud Shapiro. Programmable and autonomous computing machine made of biomolecules. *Nature*, 414:430–434, 2001.

[8] C.H. Bennett. The thermodynamics of computation – a review. *International Journal of Theoretical Physics*, 21:905–940, 1982.

[9] Ravinderjit S. Braich, Nickolas Chelyapov, Cliff Johnson, Paul W.K. Rothemund, and Leonard Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296:499–502, 2002.

[10] Dennis Bray. Protein molecules as computational elements in living cells. *Nature*, 376:307–312, 1995.

[11] K.J. Breslauer, R. Frank, H. Blocker, and L.A. Marky. Predicting DNA duplex stability from the base sequence. *Proc. Natl. Acad. Sci.*, pages 3746–3750, 1986.

[12] T.A. Brown. *Genetics: A Molecular Approach.* Chapman and Hall, 1993.

[13] Martin Campbell-Kelly and William Aspray. *Computer: A History of the Information Machine.* Westview Press, Colorado, second edition, 2004.

[14] Michael Conrad. On design principles for a molecular computer. *Communications of the ACM*, 28:464–480, 1985.

[15] Michael Conrad and E.A. Liberman. Molecular computing as a link between biological and physical theory. *Journal of Theoretical Biology*, 98:239–252, 1982.

[16] Stephen Cook. The complexity of theorem proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[17] Dirk Faulhammer, Anthony R. Cukras, Richard J. Lipton, and Laura F. Landweber. Molecular computation: RNA solutions to chess problems. *Proceedings of the National Academies of Science*, 97(4):1385–1389, 2000.

[18] Richard P. Feynman. There's plenty of room at the bottom. In D. Gilbert, editor, *Miniaturization*, pages 282–296. Reinhold, 1961.

[19] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

[20] A.M. Gibbons. *Algorithmic Graph Theory.* Cambridge University Press, 1985.

[21] Frank Guarnieri, Makiko Fliss, and Carter Bancroft. Making DNA add. *Science*, 273:220–223, 1996.

[22] Juris Hartmanis. On the weight of computations. *Bulletin of the European Association for Theoretical Computer Science*, 55:136–138, 1995.

[23] A. Hjelmfelt, F.W. Schneider, and J. Ross. Pattern recognition in coupled chemical kinetic systems. *Science*, pages 335–337, 1993.

[24] Allen Hjelmfelt, Edward D. Weinberger, and John Ross. Chemical implementation of neural networks and Turing machines. *Proceedings of the National Academy of Sciences*, 88:10983–10987, 1991.

[25] Laura F. Landweber and Eric B. Baum, editors. *Second Annual Workshop on DNA Based Computers*, Princeton University, NJ, USA, June 10–12 1996. American Mathematical Society.

[26] Richard J. Lipton. DNA solution of hard computational problems. *Science*, 268:542–545, 1995.

[27] Qinghua Liu, Liman Wang, Anthony G. Frutos, Anne E. Condon, Robert M. Corn, and Lloyd M. Smith. DNA computing on surfaces. *Nature*, 403:175–179, 2000.

[28] Chengde Mao, Thomas H. LaBean, John H. Reif, and Nadrian C. Seeman. Logical computation using algorithmic self-assembly of DNA triple-crossover molecules. *Nature*, 407:493–496, 2000.

[29] Kary B. Mullis, François Ferré, and Richard A. Gibbs, editors. *The Polymerase Chain Reaction.* Birkhauser, 1994.

[30] Mitsunori Ogihara and Animesh Ray. DNA computing on a chip. *Nature*, 403:143–144, 2000.

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

[32] Antonio Regalado. DNA computing. *MIT Technology Review*, January 11, 2002. http://www.technologyreview.com/articles/00/05/regalado0500.asp.

[33] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public key cryptosystems. *Comm. ACM*, 21:120–126, 1978.

[34] Paul W.K. Rothemund. Folding DNA to create nanoscale patterns. *Nature*, 440:297–302, 2006.

[35] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W.K. Rothemund, and Leonard M. Adleman. A sticker based architecture for DNA computation. In Landweber and Baum [25].

[36] Kensaku Sakamoto, Hidetaka Gouzu, Ken Komiya, Daisuke Kiga, Shigeyuki Yokoyama, Takashi Yokomori, and Masami Hagiya. Molecular computation by DNA hairpin formation. *Science*, 288:1223–1226, 2000.

[37] Eric Smalley. Interview with Ned Seeman. *Technology Research News*, May 4, 2005.

[38] Lloyd M. Smith. Nanostructures: The manifold faces of DNA. *Nature*, 440:283–284, 2006.

[39] H. Stubbe. *History of Genetics—from Prehistoric times to the Rediscovery of Mendel's Laws.* MIT Press, 1972.

[40] Danny van Noort, Frank-Ulrich Gast, and John S. McCaskill. DNA computing in microreactors. In Natasa Jonoska and Nadrian C. Seeman, editors, *DNA Computing: 7th International Workshop on DNA-Based Computers. LNCS 2340*, pages 33–45. Springer, 2002.

[41] John J. Watkins. *Across the Board: The Mathematics of Chess Problems.* Princeton University Press, 2004.

[42] J.D. Watson and F.H.C. Crick. Genetical implications of the structure of deoxyribose nucleic acid. *Nature*, 171:964, 1953.

[43] J.D. Watson and F.H.C. Crick. Molecular structure of nucleic acids: a structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.

[44] J.D. Watson, N.H. Hopkins, J.W. Roberts, J.A. Steitz, and A.M. Weiner. *Molecular Biology of the Gene.* BenjaminCummings, fourth edition, 1987.

[45] E. Winfree, F. Liu, L. Wenzler, and N.C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.

[46] Erik Winfree. *Algorithmic self-assembly of DNA.* PhD thesis, California Institute of Technology, May 1998.

[47] Hao Yan, Sung Ha Park, Gleb Finkelstein, John H. Reif, and Thomas H. LaBean. DNA-templated self-assembly of protein arrays and highly conductive nanowires. *Science*, 301:1882–1884, 2003.

Books and Reviews

[48] Leonard Adleman. Computing with DNA. *Scientific American*, 279(2):54–61, August 1998.

[49] Martyn Amos. *Genesis Machines: The New Science of Biocomputing.* Atlantic Books, 2006.

[50] Nancy Forbes. *Imitation of Life: How Biology is Inspiring Computing.* MIT Press, 2004.

[51] Larry Gonick and Mark Wheelis. *The Cartoon Guide to Genetics.* HarperPerennial, 1983.

[52] Richard Jones. *Soft Machines: Nanotechnology and Life.* Oxford University Press, 2004.

[53] Gheorghe Păun, Grzegorz Rozenberg, and Arto Salomaa. *DNA Computing: New Computing Paradigms.* Springer, 1998.

[54] Robert Pool. A boom in plans for DNA computing. *Science*, 268:498–499, 1995.

[55] James Watson. *DNA: The Secret of Life.* Arrow Books, 2004.