Molecular computing: **Does DNA compute?**

Daniel E. Rozen*, Steve McGrew† and Andrew D. Ellington‡

The demonstration that DNA molecules can act as parallel processors to solve hard problems has excited interest in the possibility of developing molecular computers based on recombinant DNA techniques.

Addresses: *Department of Biology, Indiana University, Bloomington, Indiana 47405, USA. †New Light Industries Ltd, 3610 S. Harrison Road, Spokane, Washington 99204, USA. ‡Department of Chemistry, Indiana University Bloomington, Indiana 47405, USA

Current Biology 1996, Vol 6 No 3:254–257

 ISSN 0960-9822

The concise algorithm by which protein functions are encoded in DNA sequences, and the flexible and adaptable operating systems of biological organisms, have long been coveted by computer scientists. Similarly, the information storage capacity of computers and their extraordinary problem-solving speed have been the envy of biologists. The parallels between the two systems are many, and it would seem natural to try to combine them. Indeed, computer scientists have been quick to take advantage of many of the information-handling paradigms that have been tried and adopted by evolution. Neural nets, genetic algorithms and cellular automata all attempt to reproduce the elegance of biological systems in silicon. But despite the fact that the binary logic of machine code is superficially similar to the quaternary logic of base-pairing, reproducing the speed of computers in carbon has until recently been unattainable.

In late 1994, however, Leonard Adleman published a paper [1] that fostered a conceptual revolution. Adleman showed how a computationally 'hard' problem could be solved using techniques from molecular biology. The most spectacular aspect of this work was the demonstration that molecules could be used to solve computational problems that require the brute-force assessment of all possible answers. As marvelous as conventional computers are at sequentially deriving answers in a limited number of steps, they become woefully slow when asked to sample and resample billions of possible answers. In contrast, molecules have evolved to pick out compatible chemical surfaces from extremely complex mixtures in a time frame that is limited only by diffusion. While conventional computers attack problems *via* mind-numbingly large calculations in series, properly encoded 'molecular computers' might quickly solve the same problems by simultaneously carrying out billions of operations in parallel.

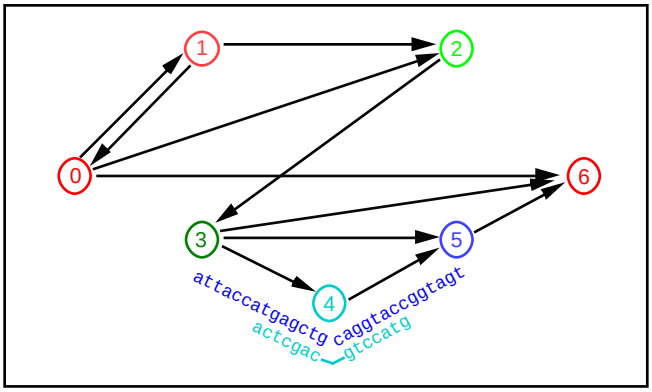
Adleman's seminal work was quickly followed by a flurry of proposals and predictions. Some researchers expounded on either the generality or shortcomings of the original DNA computer, while others attempted to expand the equivalence between bit and base by showing how DNA could be used to encode the 'central dogma' of computer science, the Turing machine. As yet, the only experimental attempt to produce a molecular computer has been by Adleman. Nonetheless, the World Wide Web contains numerous pages by DNA computer aficionados that propose eventual forays into wetware. Most of these proposals are wildly impractical. At this stage, it is difficult to predict whether the excitement over molecular computers heralds a new era in technology or will degenerate into a series of clever gimmicks and tricks. We shall attempt to summarize both published and Web-accessible documents about molecular computation, and to show that there exists great potential for additional collaboration between two of the most active fields in science.

Parallel computation and Hamiltonian paths

Adelman [1] tackled what is known as a Hamiltonian path problem, the task being to find a path through a directed graph that starts and ends at defined nodes and visits each node exactly once (Fig. 1). Adleman set out to identify a seven-node Hamiltonian path using some of the basic tools of molecular biology: oligonucleotide synthesis, the polymerase chain reaction (PCR) and affinity purification. The nodes were each represented by a 20-base DNA oligonucleotide, and the routes between them by DNA oligonucleotides complementary to the 5′ end of one node and the 3′ end of another. For simplicity, a contrived graph was used with one unique Hamiltonian path. When all of the node and route oligonucleotides were mixed together they annealed, forming double-stranded DNAs of varying lengths. Each double-stranded DNA segment outlined a particular path (not necessarily Hamiltonian). The annealed oligonucleotides were then ligated together to 'fix' each path. Paths that began and ended at particular nodes were amplified from the mixture by using oligonucleotides corresponding (or complementary) to those nodes as PCR primers.

The Hamiltonian path was verified in several ways. First, as the selected path should contain all of the nodes, only oligonucleotides that were at least as long as all of the nodes strung together were eluted from an agarose gel. Second, the Hamiltonian path should contain each of the nodes: oligonucleotides complementary to each of the different nodes were attached to affinity columns and used to successively isolate those paths of correct length that contained each node. For example, the complement of node 1 was used to isolate oligonucleotides containing node 1; these paths were then passed over a column that

Figure 1



The diagram represents a directed Hamiltonian path problem similar to the one solved by Adleman [1]. Of the many possible routes between each of the nodes, starting at 0 and ending at 6, the only one that correctly solves the Hamiltonian path problem is: 0→1→2→3→4→5→6. Adleman encoded the routes between cities/nodes as random oligomers and encoded the cities/nodes themselves as the 3′ complement of half of one route and the 5′ complement of half of the next route. The routes (shown in deep blue) are splinted together by the node (in lighter blue) that represents node 4. Such splinting through the entire graph allows a double stranded DNA encoding of the correct solution to the Hamiltonian path problem.

contained the complement of node 2, and so forth. Final verification of the Hamiltonian path was obtained by carrying out PCR reactions with node 0 as one primer and the complement of each other node, in turn, as the other primer. As there was only one unique Hamiltonian path on the contrived graph, only the full-length ligated oligonucleotide yielded PCR products of predictable size. That is, PCR primers for 'node 0 plus complement of node 1' yielded a fragment corresponding in size to 0→1, PCR primers for 'node 0 plus complement of node 2' yielded a fragment corresponding in size to 0→1→2, and so forth.

Technically, Adleman's experiment was beautifully executed. But the methods used may prove impractical for solving larger problems. Lineal and Lineal [2] note that the complexity of solving a Hamiltonian path problem increases exponentially as the number of vertices increase. Thus, for nodes of length 20 an 'n' node path would require approximately $20n(\log n)^n$ base pairs of input, and a path that was an order of magnitude greater than Adleman's (70 nodes) would therefore require "the total mass of nucleotides involved in the experiment to reach 1025 kilograms of DNA!" Stemmer [3], however, has suggested that recursive methods might be used to reduce the quantities of DNA required, although the accuracy of the solution might be sacrificed. Other resources may also be limiting. Lo, Yiu and Wong [4] suggest that a 25-node path would require prohibitive quantities of enzymes and could not possibly be analyzed by agarose gel electrophoresis.

Other problems may stem from the inherent incongruity between error-prone biology and errorless computation. For example, using PCR to regenerate the paths presents potential difficulties because the fidelity of the polymerases used in the amplification is not absolute. Similarly, primer and path annealing depend strongly upon reaction conditions, such as temperature, pH and reaction volume, which can vary from experiment to experiment. These imperfections may lead to PCR artifacts not predicted by the graph, and such artifacts may overtake and obscure real answers. Lastly, the system is restricted to reasonably short sequences, because as the sequence length increases so does the time required to obtain successful annealing. This restriction, in particular, circumvents the inherent advantages of massive molecular parallelism [5].

Adleman [6] answers these criticisms by noting that, although molecular computers are currently limited in their applicabilities, they are still 'embryonic.' He further suggests that, as the field turns from conceptualization to engineering practice, researchers will necessarily begin to address a number of important issues. First, what questions can molecular computers best answer? Second, which of the many tools used by molecular biologists can best promote molecular computation? And third, how can the architecture of a molecular computer best be designed?

The first of these goals, defining appropriate questions for molecular computers, has already begun to be addressed. Following in the path of Adleman's experiment, Richard Lipton [7] proposed a series of experiments in DNA-based computation to tackle satisfaction (SAT) problems in computer science. The aim of a SAT problem is to find a resolution to equations of AND and OR functions. Lipton shows how it should be possible to use molecular computers to solve the equation $F = (x \text{ OR } y) \text{ AND } (\bar{x} \text{ OR } \bar{y})$. The variables x and y assume Boolean values of 0 or 1 ('false' or 'true'); $\bar{x}$ and $\bar{y}$ are the negations of x and y, respectively, so that when x is 0, $\bar{x}$ is 1, and *vice versa*. The problem is to find values for x and y that make the overall proposition, F, true. To solve this problem, Lipton proposes to encode binary digits in oligonucleotides. Whereas Adleman essentially wrote arbitrary names of nodes or routes onto oligonucleotides, Lipton's scheme allows each of the answers to a SAT problem to be directly encoded. This difference is similar to the difference between a computer language with symbolic constructs and a machine language of 0s and 1s.

Lipton's conceptual advance may allow the solutions to SAT problems to be directly extracted from oligonucleotide mixtures. For the above equation for example, a series of oligonucleotides would be constructed that had values for x at one position combined with values for y at a different position. For the problem at hand, there would be four different oligonucleotides in the original mixture, with sequences corresponding to the values (0,0), (0,1), (1,0) and

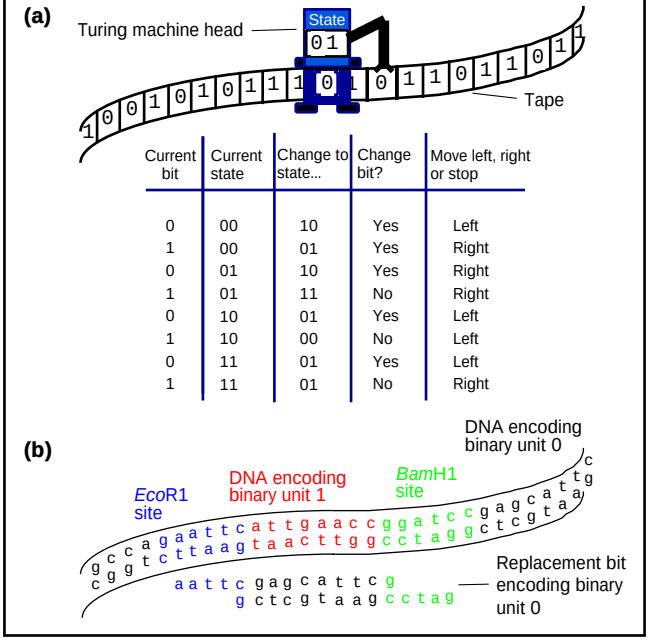
(1,1). To satisfy the first clause of the SAT problem, those oligonucleotides that had either x =1 or y = 1 would first be extracted using appropriate complementary oligonucleotides. The extracted oligonucleotides would include those whose encoded values were (0,1), (1,0) and (1,1). From this set, those oligonucleotides that satisfied the second clause of the SAT problem — sequences with either $\bar{x}$ =1 or $\bar{y}$ = 1 — would be extracted. The final, selected set would include only the oligonucleotides whose encoded values were (0,1) and (1,0). These are the correct answers to this particular SAT problem. More generally, the extraction of answers relies upon the presence or absence of a particular DNA code at a particular DNA position. Thus, after constructing a pool of all possible encoded paths, those sequences that correspond to the conditions of the equation are simply extracted.

As is the case for Hamiltonian path problems, computers solve SAT problems by iteratively searching an enormous pool of possible answers for a satisfactory answer. Once again, the massively parallel nature of molecular computers may accelerate the solution of such problems. Lipton notes that the best available methods to solve a SAT require testing 2^n possible solutions, where n is the number of variables (in the above example, n = 2). In contrast, molecular computers might solve such problems in a number of operations that increases linearly (rather than exponentially) with increases in n. To satisfy part one of an equation of a SAT problem such as the equation above, you could extract all paths that had a 1 (for 'true') in the first position. You could then further extract from that tube the members that had the desired coding in the second position, and so forth. Although no real SAT problem has yet been solved by a DNA computer, this conceptual breakthrough could allow problems whose solution sets expand exponentially ('NP-complete' problems) to be attempted. For example, Lipton and colleagues [8] have proposed a scheme for 'breaking' the data encryption standard, the algorithm underlying codes for banking and military secrecy, which relies on the fact that breaking NP-complete problems by conventional computational methods requires computational resources that are too vast to be practical. Of course, Lipton's proposal still suffers from the same real-world difficulties that limit the potential applicability of Adleman's experiments.

DNA as a Turing machine

Neither Adleman's nor Lipton's 'computer' is programmable in the traditional sense, in that they are not able to solve any problem for which there is a programmable algorithm [9]. Although the DNA oligonucleotides produced for one Hamiltonian path problem can potentially be used to solve a similar Hamiltonian path problem, that is the limit of their versatility. A Turing machine, in contrast, is the simplest programmable general model of computation, and can be described by just three essential components (Fig. 2a): an infinite information tape, a head that accesses

Figure 2



(a) A Turing machine, showing its three main features: an information tape, a machine head that reads the tape, and a table that determines the action of the machine head according to its state and the information on the tape. (b) A Turing machine implemented in DNA. Specific restriction sites surround a binary encoded oligomer that can be removed with appropriate restriction enzymes and then replaced with a segment bearing complementary sticky ends and an altered interior binary unit.

(reads and writes) the tape, and a controller that makes deterministic decisions according to a finite set of rules (a transition table). The tape is essentially a series of memory locations that each stores a single piece of information from a defined set of symbols. The head sits atop one memory location at a time and performs a complex operation: reading what is in the memory location; editing the information in the memory location according to a set of instructions in the controller; and moving left or right, or halting, depending on the state of the controller. The state of the controller changes in a deterministic way during each operation, depending on which symbol was read from the memory and the current state of the head. For each memory location and controller state, a transition table encodes a specific operation that instructs the head to change the tape symbol, change the controller state and then move to an adjacent memory location or halt.

Although no DNA Turing machine has yet been built, several possible implementations have been proposed (Fig. 2b). Beaver [10] has pointed out that text-dependent recognition, deletion and insertion can be implemented by site-directed mutagenesis, and might be used to make a DNA molecule act as a Turing machine. Rothemund [9]

describes a DNA Turing machine that might operate by recognizing and changing single bases in a DNA sequence, which could be implemented with existing restriction enzymes. His scheme uses DNA sequences (as opposed to single bases) to encode symbols, and the spacing between a recognition site and current symbol encodes the state of the Turing machine read-write head.

Winfree [11] describes a spatial Turing machine based on a self-assembling network of oligonucleotides. His notion is that an oligonucleotide with two distinct ends will tend to bind to other, complementary oligonucleotides. The binding affinities between the ends of a fairly small set of two-ended oligonucleotides would represent a set of logical operations. A series of oligonucleotides immobilized on a surface would represent an initial combination of data memory and program memory. A given program would be executed by allowing oligonucleotides in solution to bind to the first row in the initial array, then the second row and so forth. The network of annealed oligonucleotides would be equivalent to a complete record of each step in a parallel computation, such as the examples given by Adleman and Lipton.

From a practical point of view, some of the same problems exist for DNA Turing machines as for Adleman's scheme: a huge amount of material would be required to encode memory locations, the state of the controller and the transition table. Furthermore, DNA Turing machines would operate at least partially in series rather than completely in parallel, obviating some of the potential advantages of DNA computers. Electronic Turing machines are intrinsically inefficient compared to more specialized machines and algorithms, and their DNA counterparts would be likely to prove to be even slower than the electronic systems. Although molecular computers may allow computing time to scale linearly, rather than exponentially, with problem size, they may also trade a fixed machine volume for an enormous liquid volume. For example, Rothemund [9] has estimated that his version of the Turing machine would require kilograms of DNA with a fluid memory volume "about 1/10th the volume of an Olympic sized swimming pool"! Needless to say, it is unlikely that such a volume would be amenable to the manipulations required to run a molecular Turing machine.

Conclusions

Despite the numerous problems associated with the above proposals, the field of molecular computation has a promising future. The problems associated with huge matrices, genetic algorithms, optimization schemes or database searches are quite real. While there may not be a need to solve a 1000-node Hamiltonian path problem, embedding logical elements in DNA will stimulate both molecular biologists and computer scientists to produce new and better ideas. Researchers in both fields have already been forced to re-examine what elements define computation. For example, Bunow [12] and Reif [5] have stated that the techniques of *in vitro* evolution [13], wherein on the order of 10^{13} DNA or RNA molecules are screened simultaneously for some ligand-binding or catalytic activity, is a form of computation, a claim that biologists themselves would probably not have made.

Considering that Adleman's paper is less than two years old, enormous conceptual progress has already been made. The true problems, however, lie on the technical front. Smith [14] notes that there has been a "severe deficit of actual in-the-lab biochem types" working and collaborating to develop realistic models of molecular computation. In particular, molecular biologists must focus on determining what systems are most amenable to molecular computation, and what computations are most amenable to molecules or organisms. For example, it may be possible to encode simple logical elements, such as binary counters or limited transition tables, in plasmids or other vectors. These vectors would then make decisions about whether to turn genes on or off based on the environments they had seen and how those environments modified encoded logical elements. Such schemes would not differ greatly from those already used by biology to regulate gene expression, but would have the advantage that they could be programmed for uniquely biotechnological purposes. As collaborations between molecular biologists and computer scientists continue to evolve, so will the science they produce.

References

1. Adleman LM: Molecular computation of solutions to combinatorial problems. *Science* 1994, 266:1021–1024.
2. Linial M, Linial N: On the potential of molecular computing. *Science* 1995, 268:481.
3. Stemmer WPC: The evolution of molecular computation. *Science* 1995, 270:1510.
4. Lo YMD, Yiu KFC, Wong SL: On the potential of molecular computing. *Science* 1995, 268: 482.
5. Reif JH: Parallel molecular computation: models and simulations. Available at: http://www.cs.duke.edu/reif.HomePage.html.
6. Adleman LM: On the potential of molecular computing. *Science* 1995, 268:483–484
7. Lipton RJ: DNA solution of hard computational problems. *Science* 1995, 268:542–545.
8. Boneh D, Dunworth C, Lipton R: Breaking DES using a molecular computer. Available at: http://www.CS.Princeton.EDU/dabo.biocomp.html.
9. Rothemund PWK: A DNA and restriction enzyme implementation of Turing Machines. Available at: http://www.ugcs.caltech.edu/~pwkr/oett.html.
10. Beaver D: Molecular computing. available at: http://www.cse.psu.edu/~beaver/publications/tm.ps.z.
11. Winfree E: On the computational power of DNA annealing and ligation. Available at http://dope.caltech.edu/winfree/papers/ligation.ps.
12. Bunow B: On the potential of molecular computing. *Science* 1995, 268:482–483.
13. Bartel DP, Szostak JW: Isolation of new ribozymes from a large pool of random sequences. *Science* 1993: 261:1411–1418
14. Smith WD: Summary of the Mini DIMACS workshop on DNA-based computers. Available at: http://www.neci..nj.nec.com/homepages/smith/workshop.summary.ps.