Biomolecular Computing and Programming

Max H. Garzon and **Russell J. Deaton**
and The Molecular Computing Group
http://www.msci.memphis.edu/~garzonm/mcg.html
The University of Memphis, Memphis TN, 38152

Abstract

Molecular computing is a discipline that aims at harnessing individual molecules at nanoscales for computational purposes. The best studied molecules for this purpose to date have been DNA and bacteriorhodopsin. Molecular computing allows one to realistically entertain, for the first time in history, the possibility of exploiting the massive parallelism at nanoscales inherent in natural phenomena to solve computational problems. The implementation of evolutionary algorithms in biomolecules would bring about full circle the biological analogy and present an atractive alternative to meet large demands on computational power. This paper presents a review of the most important advances in molecular computing in the last few years. Major achievements to date are outlined, both experimental and theoretical, and major potential advances and challenges for practicioners in the foreseeable future are identified. A list of sources and major events in the field has been compiled in an appendix, although no exhaustive survey of the expanding literature is intended.

1 Introduction

The notion of harnessing individual molecules at nanoscales for computational purposes is an idea that can be traced back at least to the time when electronic computers were being constructed in the 1940s. Electrons are, in fact, orders of magnitude smaller than molecules, but over 10^{15} are required just to communicate a carriage return to a conventional processor. The idea of improving the efficiency of hardware utilization using biomolecules is attractive for several reasons. First, hardware is inherently parallel, and parallelism is a good way to handle computational bottlenecks. Second, biomolecules occur abundantly in nature, for example, inside all known living cells with (eukaryote) and without (prokaryote) nuclei, and constitute the basic substratum of life. Consequently, they have developed a structure that enables them to solve a number of difficulties for parallel computing, such as communication and load balancing problems, by mechanisms that we may not even be aware of. Furthermore, short biomolecules can now be synthesized at low costs. Third, their physical implementation is therefore relatively simple compared to the demanding and costly fabrication processes used in VLSI. Let us consider the following figures in terms of sheer space, not to mention performance. A human brain consists of about 10^{12} neurons and a human body comprises over 10^{15} cells; each cell contains a copy of the entire genetic code consisting of over 3 billion nucleotide pairs to perform living functions, all that nicely packed in a few double helices about 3.4 nanometers wide and microns long. Therefore, computing based on molecular media would really tip the scales in terms of miniaturization. On the other hand, these advantages are obtained at the expense of complications that are nonissues for conventional computers, as will be seen presently. The basic problem for computation remains: how to trick a piece of matter (biomolecules in this case), evolved to have a "mind of its own" following predetermined physical and/or chemical laws, to perform an anthropomorphic task typical of what we understand today as computation?

The purpose of this article is to present an review of the most important advances in molecular computing in the last few years, and to identify some of the great challenges for the field in the foreseeable future. In the process, we identify some of the potential advances that the field may make. Section 2 contains technical details of Adleman's landmark experiment as well as a brief review of some of the origins of DNA computing. Section 3 outlines some of the major achievements to date. Section 4 identifies what we believe are the major challenges if the field is to eventually become an established paradigm with real-life applications. They originate in projects that have been suggested or are in progress, and so may generate major breakthroughs in the next few years. Finally, in Section 5 we make a summary assessment of achievement and outlook for the future.

A list of sources and major events in the field have been compiled in an Appendix, but we have certainly not attempted to make an exhaustive survey of the literature. The selection and emphasis of the topics are representative of the main directions in the field, but also undoubtedly reflect some of the authors' biases.

2 The Origins of Molecular Computing

Lately, advances in computer science have been characterized by the computational implementation of well-established biological paradigms. Notable advances are artificial neural nets inspired by the brain and its obvious connection to natural intelligence, and evolutionary computation, inspired by the Darwinian paradigm of natural selection. Early ideas of molecular computing attempted to emulate conventional electronic implementations in other media, e.g., implementing Boolean gates in a variety of ways. A fundamental breakthrough characteristic of a new era was made by Adleman's 1994 paper [1], where he reports an experiment performed with molecules of fundamental importance for life, DNA (deoxyribonucleic acid) molecules, to solve a computational problem known to be difficult for ordinary computers, namely the HAMILTONIAN PATH PROBLEM (HPP). This problem is typical of an elite set of problems in the well-known complexity class **NP** that exemplify the computational difficulty of search procedures that plague a number of very important applications in combinatorial optimization, operations research, and numerical computation. Adleman's experiment ushered in a new computational paradigm in molecular computing for several reasons. First, it showed that it is indeed possible to orchestrate individual molecules to perform computational tasks. Second, it showed the enormous potential of DNA molecules for solving problems beyond the reach of conventional computers that have been or may be developed in the future based on solid-state electronics. Shortly after, the first conference on DNA-based computing was organized at Princeton University in 1995, and several events have been held since annually. (See the Appendix for a listing.)

2.1 Adleman's Landmark Experiment

In this section we present the essential technical details of Adleman's experiment. The HPP is defined precisely as follows:

HAMILTONIAN PATH PROBLEM (HPP)

Instance: a directed graph Γ and two vertices, source and destination.

Question: **yes/no**, there is a path following arcs in the graph connecting the source to the destination vertices and passing through each other vertex exactly once.

As mentioned before, this problem is **NP**-complete, i.e., it is representative of many of the difficulties that afflict conventional computers for solving very important problems in combinatorial optimization and operations research. Each complete problem in **NP** contains all problems in the class **NP** as special cases, after some rewording, and is characterized by the fact that their solutions are easily verifiable, but extremely difficult to find in a reasonable amount of search time. (More technical details about this class can be found in [2].) The best-known general techniques to apply to these problems amount essentially to an exhaustive search through all possible solutions, looking for satisfaction of the constraints required by the problem. It is therefore an ideal candidate for a brand new computational approach using molecules.

Adleman's brilliant insight was to carefully arrange a set of DNA molecules so that the chemistry that they naturally follow would perform the brunt of the computational process. The key operations in this chemistry are sticking operations that allow the basic nucleotides of nucleic acids to form larger structures through the processes of *ligation* and *hybridization* (more below in Section 3.1). The first DNA-based molecular computation is summarized in Fig. 1. Specifically, Adleman assigned well chosen unique single-stranded molecules to represent the vertices, used Watson-Crick complements of the corresponding halves to represent edges joining two vertices, and synthesized a picomol of each of the 21 resulting molecules for the graph in Fig. 1(a). Taking advantage of the fact that molecular biologists have developed an impressive array of technology to manipulate DNA, he designed a molecular protocol (one would say *algorithm* in computer science) that enabled the molecules to stick together in essentially all possible ways. In the situation illustrated in Fig. 1(b), the edge molecules were to splinter nearby vertex molecules to construct longer and longer molecules representing paths in the original graph. If there exists a Hamiltonian path called for in the problem specification, one representative molecule would thus be created by the chemistry on its way to equilibrium. Using more of the same biotechnology he could then determine, as illustrated in Fig. 1(c), the presence or absence of the molecule in the final test tube and respond accordingly to the original problem. (Some details of these operations can be found below in Section 3.1). The full technical details of the reactions and experimental setup can be

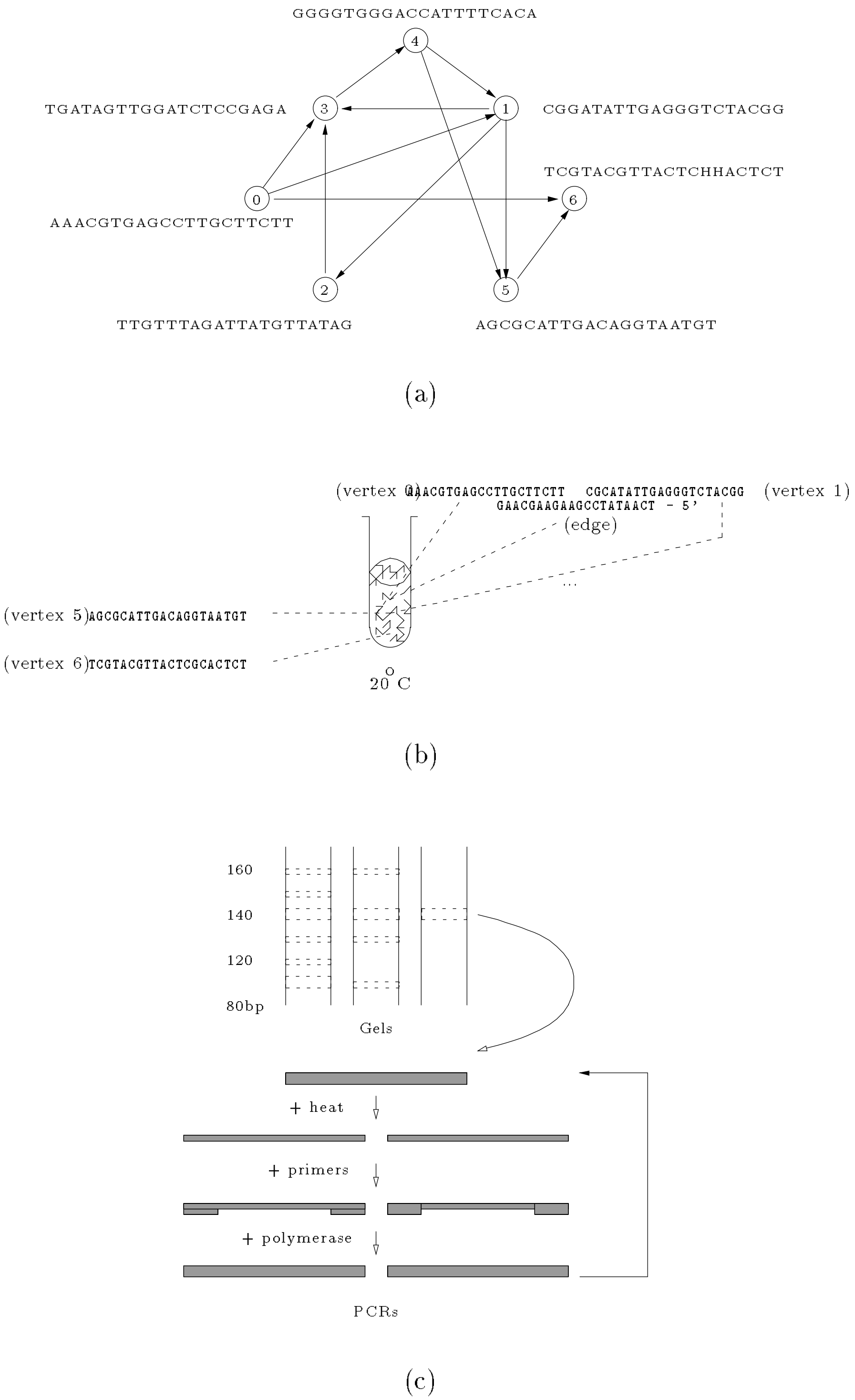


Figure 1: Steps in Adleman's Molecular Computation: (a) encoding of problem instance; (b) computing reactions; (c) extraction.

found in [1].

This paper provided a very appealing argument for molecular computing. In addition to the points mentioned at the end of the previous section, the most important point is perhaps that biotechnology is mature enough to stop dreaming about gedanken experiments for solving hard computational problems, and that it is time to begin thinking about specific experimental setups to solve them.

2.2 DNA Computation in Ciliates

Four years later, Landweber and Kari [3] presented a different version on the origin of DNA computing. They provide a convincing argument that several million years earlier and unknown to Adleman, the ciliated protozoa *oxytricha nova* and *oxytricha trifallax* of the genus *oxytricha* solved a problem similar to HPP while unscrambling genes as part of their reproductive cycle. Ciliate cells possess, in general, two nuclei, an active macronucleus and a functionally inert micronucleus. The macronucleus forms from the micronucleus after sexual reproduction. The process requires more than simple copying, however, because intervening nonprotein coding sequences that shatter it into nearly a hundred pieces must be removed, and moreover, the relevant protein-coding sequences sometimes appear scrambled and must be restored to their natural order. This process is essentially identical to the problem one faces in HPP , namely to arrange the cities in the right order for a Hamiltonian path. The analogy goes further since the protozoa seem to rely on short repeat sequences that act as sort of matching tags in recombination events. If the mechanisms underlying this type of information processing can be fully attributed to the same kinds of processes present in Adleman's experiment, then molecular computation is certainly millions of years old. Therefore, the origins of molecular computing are still buried in the evolution of genetic complexity in the biological kingdom.

2.3 Other substrata for Molecular Computing

Although the question has been raised several times as to whether DNA is necessarily the molecule best suited for molecular computing, only a couple of papers have addressed alternatives such as RNA (rybonucleic acid) in depth. In two recent papers [4, 5], The Princeton group has provided detailed experimental evidence that RNA may be more suitable to solve some computational problems because of its versatility. They argue that RNA is easier to use in substractive protocols that solve the problem by eliminating molecules representing problem constraint violations, rather than Adleman's additive method of building up the solution from basic building strands. Specific advantages will be sprinkled throughout the rest of this paper.

An older alternative to DNA molecules that support optical computing is the protein *bacteriorhodopsin*, which contains the light sensitive *rhodopsin* present in vertebrate retinas. In essence, this molecule consists of seven alpha-helical segments that span the purple membrane of a microorganism commonly known as *halobacterium halobium*. This organism grows in salt marshes at higher salt concentrations than sea water, where exposure to high thermal fluctuations and photochemical damage has made it capable, for the sake of metabolic energy, of switching chemically among a few atomic states a thousand timefold more efficiently than similar synthetic materials. Switching can takes place by absorption of green and blue light as many as 10 million times before wearing out. The switching property has been used in combination with lasers to create a storage medium for optical computer memories that is almost in the commercial stage now. The possibility exists that it might become a core memory for a molecular computer [6, 7]. Although certainly involving amino acids at the protein-binding sites, this type of computation is more passive than the type described above. We thus use the expression 'molecular computing' in order to avoid excluding any future developments with other media, but with the understanding that, currently, it essentially means DNA- and RNA-based computing. (It has been argued, in fact, that other molecules such as proteins and artificially engineered rybozymes may serve biological and computational functions much better. See Ellington et al. [8, 9] and Kool [10] for more details.)

3 Some Success Stories

In this section we give a brief description of some of the problems for which molecular protocols have been or are being implemented successfully. Each story below either illustrates a basic technique in molecular computing, or has successfully marked definite progress in the lab. Before proceeding further, however, we need to give a more precise description of the molecular biological background as well as a characterization of the basic methodology employed in molecular computing.

3.1 Basics from Molecular Biology

We present here the bare bones necessary to pin down the biochemical foundations of molecular computing. A more sophisticated reader is referred to [11, 12] for further background in molecular biology.

The relevant molecule chemistry is that of DNA and RNA. These complex molecules are composed of basic blocks called nucleotides, nucleic acid bases $\texttt{A}$, $\texttt{T}$, $\texttt{G}$, $\texttt{C}$, that bind to form chains called oligonucleotides, or $n-$mers, according to the Watson-Crick (herein abbreviated as WC) complement condition, $\overline{\texttt{A}} = \texttt{T}$ and $\overline{\texttt{C}} \equiv \texttt{G}$, and *vice versa.* Each molecule has a polarity (sense of orientation) from a so-called $5'$-end to a $3'$-end or vice versa. (The ordinary convention is to write them from the $5'$ to the $3'$ end, unless we are describing double strands, in which case the lower oligonucleotide is directed in the opposite sense.)

The basic reactions that take place between nucleotides rely on two key properties of DNA/RNA molecules. Oligonucleotides bind in an antiparallel way with respect to the chemically distinct ends, $5'$ and $3'$, of the DNA molecule through strong co-valent bonds. *Ligation* is a chemical process whereby two double strands are joined into one double strand by the same type of binding; the process is catalyzed by an enzyme called *ligase. Hybridization* is a chemical process that joins two complementary single strands into a double strand through hydrogen bonds (two for A-T and three for C-G). A *restriction enzyme* (such as SmaI or EcoRI) is a protein characterized by the double strand DNA sequence which it recognizes (called a *site*, such as $\begin{array}{c}\texttt{CCC'GGG}\\\texttt{GGG'CCC}\end{array}$ for SmaI, and $\begin{array}{c}\texttt{CTTAA'G}\\\texttt{G'AATTC}\end{array}$ for EcoRI) and cuts into two segments (ending in $\begin{array}{c}\texttt{CCC}\\\texttt{GGG}\end{array}$ and $\begin{array}{c}\texttt{GGG}\\\texttt{CCC}\end{array}$ for SmaI, and $\begin{array}{l}\texttt{CTTAA}\\\texttt{G}\end{array}$ and $\begin{array}{r}\texttt{G}\\\texttt{AATTC}\end{array}$ for EcoRI). The use of restriction sites as a computation methodology was introduced more than a decade ago by Head [13] in the form of *splicing systems.*

The basic methodology in molecular computing to solve computational problems consists of three basic steps: an *encoding* that maps the problem onto DNA strands, *hybridization/ligation* that performs the basic core processing and *extraction* that makes the results visible to the naked eye, as illustrated in Fig. 1(c). In Adleman's solution of HPP, the vertices of the graph are encoded in oligonucleotides of DNA. The encoding of an edge consists of a splicing complement of the two halves of the end vertex representations. The DNA representation of the Hamiltonian path is formed as the vertex oligonucleotides are bound by hybridizing with the edge oligonucleotides. Ideally, the Hamiltonian path, if present, is produced if the proper hybridizations occur.

Combination of ligation, hybridization, and cleaving yields other cut-and-paste operations that permit manipulation of these strands at the nanoscale levels where they occur. It is the nature of biomolecules to permit only procedures that are uncertain and statistical in nature so that results can be guaranteed only globally, not at the level of specific molecules. The most important procedures for molecular computing can be characterized as follows (we again refer the reader to [11] for the intimate biochemical details).

Gel Electrophoresis
This procedure acts as a powerful microscope that permits us to see molecules (or rather populations thereof) with the naked eye. This operation effects a sorting procedure on a population of molecules of various lengths. It exploits the fact that biomolecules have a negative electric charge, so that when placed in an electric field, they will tend to move to the anode. If they are forced to move through a resistive medium (such as a gel or thin capillaries), it will exert a drag that is inversely proportional to their length. Biomolecules also absorb ultraviolet radiation in a characteristic way, so that, after an appropriate time, a UV snapshot will show bands where a subpopulation of strands of the same length have clustered together whenever they are numerous enough to show a detectable effect.

Enzymes
Restriction enzymes play the role of scissors in cut-and-paste operations. They come in several flavors. Restriction enzymes (such as `E-coli` or `AluI`) effect cutting vertically to a double strand as described above. `Exonucleases` and `endonucleases` pluck off nucleotides from a double strand horizontally parallel to the strand. `Polymerase` grabs freely floating nucleotides in the neighborhood onto a single stranded that has been properly primed with complementary markers at two ends (in computer science one would say initialized) and creates a double-stranded segment between the primers, as illustrated in Fig. 1(b). Currently, about 200 naturally occurring enzymes have been identified. The most useful for molecular computing include the ones already mentioned and `Taq`. The artificial evolution of enzymes, more akin to ribozymes but with analogous properties, has been proposed as an alternative subject of research for molecular computing – see, for example, [14, 8].

PCR-Polymerase Chain Reaction
Introduced only in 1984, this procedure works as a copier machine for molecular computing. It allows duplicating double-stranded molecules determined by single-stranded end-markers called *primers*, as illustrated in Fig. 1(c). It consists of heating the population to be duplicated to denature (melt) the double strands into single strands; primers are added that hybridize to their WC-complements on the single strands and mark off the duplication region. The enzyme *polymerase* is added to the reaction that successively attaches complementary nucleotides on the two single-stranded segments until they become two double-stranded copies of the original molecule. Exponential amounts of single molecules can then be obtained after a few iterations of thermal cycles and enzyme addition. The process can be performed automatically for double strands of length up to 200 bp (base pairs).

There are undoutbedly other tools in the biotechnology kit, but these will suffice for the sake of this paper. These basic operations can be further assembled into basic instruction sets to arrange protocols that perform meaningful computational procedures.

3.2 Parallel Overlap Assembly

Perhaps the foremost advantage of computing with molecules is the ready parallelism in which molecular operations take place. The best way to exploit this parallelism is to perform the same operation simultaneously on many molecules. Adleman's basic technique can be characterized as a generate-and-filter technique, i.e., generate all possible paths and filter out those that are not Hamiltonian. In this approach, one must be sure to generate all the possible solutions to the problem, akin to making sure that the data structure for a chromosome captures all possible solutions in an evolutionary algorithm. Many protocols in molecular computing exploit this method, for example Boolean formula evaluation (see the next section for some references).

It is therefore important to be able to generate all potential solutions to a problem. A procedure called *parallel overlap assembly* has been used in molecular biology for gene reconstruction and DNA shuffling. It has been successfully used by Ouyang et al. [15] in a lab experiment to solve an instance of another **NP**-complete graph problem, MAX-CLIQUE. The procedure consists of iterations of thermal cycles that anneal given shorter DNA segments in random ways to produce larger potential solutions. Related procedures have been used to improve solutions to HPP by Arita et al. [16].

3.3 Boolean-circuit Evaluation

Another important approach in molecular computing is the implementation of Boolean circuits because that would allow importing to the world of molecules the vast progress that has been made on information processing in electronic computers. A successful implementation of Boolean circuits would lead to the construction of ordinary computers in biomolecules, particularly of parallel computers. Lipton [17] presented an early proposal for Boolean circuit evaluation as a solution to SAT (Boolean formula satisfiability) and thereby problems in the class NP. Ogihara and Ray [18] have suggested protocols for implementing Boolean circuits that run in an amount of time that is proportional to the size of the circuit. Amos et al. [19] improved the implementation to have run time that is proportional to the depth of the circuit. In the protocol suggested by the latter, for example, input bits are represented by well-chosen $l-$mers x and y that are present in the tube if and only if the corresponding Boolean variables have value 1. The gates are represented by $3l-$mers $\bar{x}\bar{y}z$ that contain segments that are complementary to the input molecules and the output of the gate. (Without loss of generality one can assume all gates are simply NAND-gates since this operator is logically complete.) A typical evaluation of a NAND is made by placing in a tube the values of the inputs equal to 1 and allowing the formation of a double strand that represents the evaluation, as illustrated in Fig. 2(a). The encodings will have been chosen so that this evaluation will contain a restriction site for an enzyme that will destroy the molecule in case both inputs are present, so the evaluation will be 0, as expected. Otherwise, the site is used to detach the resulting value 1, which is then recursively fed into the evaluation of the next layer. This implementation allows parallel evaluation of all gates in the same layer, and so takes place in a number of steps proportional to the depth of the circuit.

A different and more versatile implementation by Hagiya and Arita [20] can be used for a similar purpose, but it has other interesting applications shown next.

3.4 Whiplash PCR

Another technique that appears to be general enough to have a good number of important applications has been implemented by Hagiya et al. [22] and termed Whiplash PCR by Adleman. The applications include ready solutions to CNF-SAT (conjunctive normal form satisfiability), DIRECT SUM COVER, and HPP [23]. An important characterization of the computational power (branching programs) together with a generalization of the technique (GO-TO programs) were later found by Winfree [24]. The protocol works for computational tasks that can be specified by state transitions. Evaluation of a circuit is an example, where the state of the computation is initially the vector

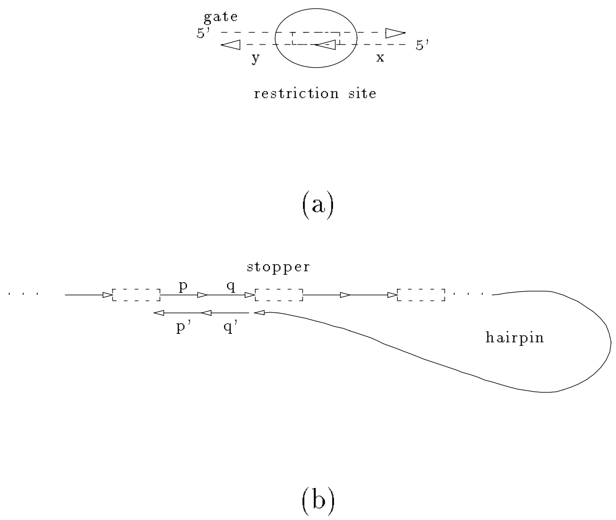


Figure 2: Two Protocols in Molecular Computation: (a) Boolean circuit implementation; (b) whiplash PCR.

of input variables, and a transition produces the values of the next layer of gates evaluated on the previous state. There are several notable features in this protocol. First, its implementation is reduced to thermal cycles that are easily automated while performing a large number of state transitions in parallel. Second, the transitions are self-controlled, i.e., once initiated, they stop themselves without explicit human intervention. Third, and more importantly, the protocol has been tested in the lab and it appears to operate as desired with high realiability, at least for a few transitions.

The protocol is implemented on single strands and exploits hairpins, i.e., the tendency of one end of a single strand to bend upon itself and hybridize to another segment of itself. The states of the computation are encoded by segments of DNA; a transition $p \to q$ on input a is encoded by the corresponding concatenation of states pq separated by stoppers whose function will be seen shortly; the $5'$-end of the strand encodes the transition table of the computation as well as its initial configuration; the current state of the computation is encoded at the $3'$-end by the complementary strand; a transition is implemented by procuring the appropriate reaction conditions (temperature cycles and polymerase) so that the current state $\bar{p}$ (denoted p' in the illustration in Fig. 2(b)) hybridizes with the appropriate transition on the $5'$-end and the polymerization doubles-strands the resulting double segment up to the stopper that separates it from the next transition.

The protocol was originally proposed as a solution to the problem of learning Boolean μ-formulas [22], which are known *not* to be learnable from examples alone in Turing polynomial time. (A μ-formula is one in which every variable occurs at most once.) Winfree [24] shows how to circumvent this restriction by introducing the linear programming equivalent of dummy variables and enforcing certain equality conditions. The hairpin, usually regarded as a source of errors, indeed appears to be a powerful tool in molecular computing.

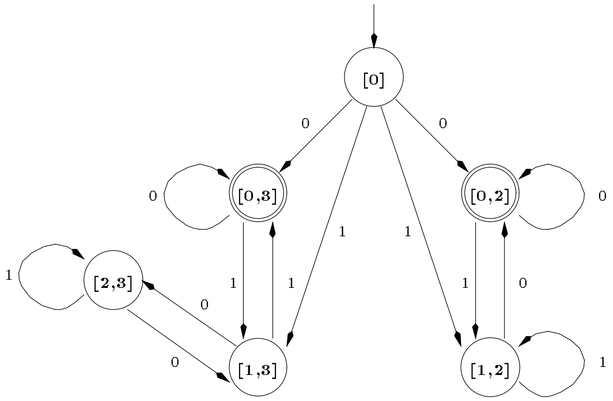


Figure 3: A Nondeterministic Finite-State Machine.

3.5 Finite-state and Turing machine implementation

There were other simultaneous attempts to implement state machines, particularly finite-state machines (FSM). Garzon et al. [25] suggested implementation of *nondeterministic* finite-state machines that are self-controlled and fault-tolerant. Fig. 3 shows a FSM where various moves are possible from a particular state **0** on the same input 0. Nondeterministic computation is at the core of the difficulties that Adleman's original experiment was designed to overcome. On the other hand, nondeterminism is supposed to be well understood in the context of finite-state machines, specifically, a so-called *subset construction* produces a deterministic equivalent of a given nondeterministic FSM. It is conceivable that greater insight about the virtues of molecular computing may be gained by looking for ways to implement nondeterminism efficiently as a native mode of computation in a fault-tolerant and efficient way.

The implementation requires a dynamic molecule to represent the changing states of the FSM that is capable of detecting its inputs in its environment. It can be a double-stranded molecule containing a segment encoding for the current state and another segment encoding the last symbol read that led to the current state. Other molecules representing inputs are added with appropriate overhangs, which upon hybridization create restriction sites that allow cleaving with appropriate enzymes to detach the old state and create a new molecule that reflects the new state. Nodeterministic transitions occur because of the various possibilities in the hybridization process. If run uncontrolled, the protocol will soon produce too many copies of the finite control in the same state and thereby thwart the efficiency of the computation. The key to the success in the subset construction to determinize a FSM is that, whenever two nondeterministic copies of the machine find themselves in the same state, one can safely discard one of them since their runs will be identical thereafter. It is desirable to have a protocol that renders the implementation efficient in the tube, i.e., that it will self-regulate to produce approximately equal concentrations of the molecules representing the various states.

The authors have proposed using the *methylation* process that occurrs in living cells and is readily implementable in the lab, described as follows. Any living cell that produces a restriction enzyme must identify and protect its own DNA from restriction. In order to prevent restriction of a cell's own DNA when the cell produces a restriction enzyme, the cell also produces a methylase enzyme (methyltransferase) which methylates, i.e., adds a $-CH_3$ chemical group to certain bases within or near the restriction sites. Nearly all restriction enzymes are thus inhibited by certain methylations of cytosine or adenine bases within their restriction sites. Common methylations include: $N4-$methylcytosine (the nitrogen at position 4 of cytosine is methylated); $C5-$methylcytosine (position 5 carbon of cytosine methylated, etc); hydroxymethylcytosine; and $N6-$methyladenine. (More can be found in [26].) This reaction can be made self-regulating by methylating a fraction of the input molecules being put into the reaction (which fraction will depend on the particulars of the NFSM being implemented). If a state is represented by a palindromic restriction site, state molecules will hybridize with other like-molecules representing the same state. The presence of the appropriate restriction enzyme in the reaction will then cut and stop further expansion of this copy of the FSM. However, if methylated bases are found within a fraction of the restriction sites formed, the restriction enzyme will be unable to cut those copies of the FSM. Thus, the constant combined presence of restriction enzymes and methylated input bases will guarantee that the number of molecules in the tube can be maintained within an appropriate range for each state, so that they will be fairly represented as the implementation proceeds. Further details of the implementations can be found in [25]. A similar more general device for a Turing machine has been proposed by Shapiro [27].

3.6 Cellular Automata Runs

A new direction originated in Winfree's attempts to show that abstract tilings, used earlier to show computational universality, can actually be implemented in the lab. He has used the XOR-rule in a 2D cellular automaton whose run on the single-input configuration 1 generates Pascal's triangle modulo 2 in order to show that it is possible to implement cellular automata in a test tube. The molecules are arranged so that the thermodynamics of the process drives the run of the automaton. The implementation has required a careful choice of three-branched and double-crossing tiles that have three or four sticky ends that hybridize to other like molecules to form a tiling. These choices will again guarantee self-regulation to ensure that only proper tiles are added at each time. Experiments have actually been performed and confirmation has been obtained that large fragments of the intended structure have been created. Naturally, other devices were required to detect these tiny structures at the nanoscales in which they exist, namely atomic-force microscopy. More details can be found in [24].

3.7 Other Applications: Is There a "killer application"?

Molecular computing has generally aimed, so far, at solving the same ordinary algorithmic problems that are commonly posed for conventional VLSI-based computers, albeit by an entirely different type of operational process. None of them have exhibited the kind of practical success that would be considered satisfactory to answer Lipton's impromptu call for a "killer app" at the second DNA-based workshop in Princeton. Such an application would suit well the nature of biomolecules, beat current and perhaps even future solid-state electronics, and would establish beyond the shadow of a doubt the power of the new computational paradigm. Landweber et al. [28] have proposed that DNA sequencing, DNA fingerprinting, and/or DNA population screening are good candidates. They would require a new approach in the way we conceive and do molecular computation now. Specifically, thus far a practicioner is assumed to know the composition (a digital string) of the molecules that initially encode information and their subsequent transformations in the tube. This methodology requires going back-and-forth between the digital and analog DNA world, by sequencing (when compositions are unknown), which is an expensive step, and the converse step, synthesis of DNA. This so-called DNA^{2}DNA approach bypasses digitizing by operating directly on unknown pieces X of DNA, using known molecules, in order to compute a predetermined function $f(X)$ that specifies the computational task.

The fact that the first five years of work in the field have not, however, produced such a killer application would make some people think that perhaps fundamental scientific and/or technological difficulties have to be overcome before one effectively appears on the scene. These proposals can be thus regarded as challenges, rather than established results and will be discussed in the following section.

4 Grand Challenges for Molecular Computing

After the discussion in the previous sections, one question must have emerged in the reader's mind: How can all this potential of molecular computing be fully realized in real life? In this section we examine fundamental challenges that will need to be resolved for bringing molecular computing to an effective new paradigm for computational science.

The root of the difficulties for molecular computing lies in our relatively poor ability to control the physical chemistry involved in the context of information processing, despite impressive progress in biotechnology that has made it thinkable. Molecular computing is based on operations performed by individual molecules. Even under the perhaps unreasonable assumption that reactions are governed by deterministic laws, the number of factors that eventually determine whether these operations take place for individual molecules is ultimately out of the control of the experimentalist, at least presently. The computation and extraction phases therefore rely on cooperative phenomena that can only be observed as *ensemble statistical processes* involving a great number of individual molecules. Early approaches simply ignored these facts. But over time, practitioners have come to realize that the future of the field may lie precisely on whether or not these problems can be overcome to a reasonable degree of effectiveness. We discuss first the more fundamental problem arising from the physico-chemical foundations because they determine to a great extent the possibilities for the more direct computer science issues to be discussed thereafter. The latter issues are worth discussing only in case good enough solutions can eventually be devised for the former.

4.1 Reliability, Efficiency, and Scalability

Reliability, efficiency, and scalability are perhaps the three most burning issues for molecular computing. The reliability of a protocol, i.e., a DNA computation, is the degree of confidence that a lab experiment provides a true answer to the given problem. The efficiency of the protocol refers to the intended and effective use of the molecules that intervene in it. The scalability of a lab experiment is the effective reproducibility of the experiment with longer molecules that can encode larger problem instances while still obtaining equally realiable results under comparable efficiency. These three are distinct but clearly interrelated problems. Biologists have not really faced these problems in their work because, in that field, the definition of success is different than in computer science. (When a biologist claims that she has cloned an organism, for example, the contention is that one experiment was successful, regardless of how many were previously not, or whether only one clone was actually produced.) Research on these problems in molecular computing has just begun. Most work has concentrated on reliability and we proceed to sketch it, in the guise of a more basic and important problem, the encoding problem. This is a good example in which molecular computing will probably have a feedback effect on the notions of efficiency and scalability in biology.

4.2 The Encoding Problem

Once the encoding molecules for the input of a problem have been chosen, a molecular computer scientist is at the mercy of the chemistry, even though she may still have some control over the protocols that she may perform with them in the laboratory execution. If the encodings are prone to errors, the experiment can be repeated any number of times and always provide the same (erroneous) results, as evidenced in [29]. This fact lessens the effectiveness of the standard method of increasing the reliability of a probabilistic computation with a nonzero probability of errors by iteration, contemplated in [30, 31]. A different analysis of the problem was initiated by Baum [32], where it is assumed that undesirable errors will occur only if repetitions or complementary substrands x of a certain minimum sticking length $k := |x|$ appeared in the encoding. The problem is that the uncertain nature of hybridizations plagues the separators that are used to prevent the problem, so a more thorough approach appears to be necessary.

A *mismatched hybridization* is a bound pair of oligonucleotides that contains at least one mismatched pair. In addition to frame shift errors in which the $n-$mers are shifted relative to each other, mismatches leading to false positives include hairpin mismatches, bulges, and partial hybridizations. The *encoding problem* for DNA computing thus consists of mapping the instances of an algorithmic problem in a systematic manner onto specific molecules so that (a) the chemical protocols avoid all these sources of error, and (b) the resulting products contain, with a *high degree of reliability*, enough molecules encoding the answers to the problem's instances to enable a successful extraction.

An optimal encoding would maximize the likelihood of desired hybridizations while minimizing the occurrence of undesirable hybridizations, and furthermore lead to equilibrium reaction conditions that are favorable for retrieving the solution of the problem in the extraction phase. Clearly, the encoding of a problem for a molecular solution has to be decided beforehand by means presumably different from DNA computation. Thus, in its full generality, we have the following algorithmic problem.

DNA ENCODING(τ)

Instance: A finite set S of $n-$mers over the genetic alphabet $\Sigma := \{\mathtt{A}, \mathtt{G}, \mathtt{C}, \mathtt{T}\}$, and a positive integer K

Question: Is there a subset $C \subseteq S$ such that

$$\forall x, y \in C, \tau(x, y) \geq K\,?$$

The function τ reflects a desirable quality criterion for the protocol and can be given by mapping $\tau : \Sigma^* \to \mathbf{Z}^+$. Solving the encoding problem requires identifying appropriate criteria that capture the relevant chemistry, and moreover, give algorithms to produce good encodings that will satisfy constraints (a) and (b).

The most natural and fitting criteria can be found in the thermodynamics that governs the hybridization and ligation processes. Ultimately, it comes down to the Gibbs free-energy that nucleotides release during hybridization in passing to a lower energy state of a bound pair. The thermodynamics of hybridizations is fairly well known – see Wetmur [33] for a survey of relevant facts, as well as SantaLucia et al. [34]. The basic quantity is the melting temperature T_m of a given double strand, which is defined as the temperature at which half of a homogenous population of such double strands will have denatured into single strands. The controlling parameters of a melting temperature are strand composition, strand concentration, and various other solvent properties such as pH of the solution. Despite some fundamental work [35, 36, 37, 38], this approach based on melting temperatures has not really produced a systematic and productive way to produce good encodings. Such encodings can actually be obtained through evolutionary searches, either in vitro [39, 40] or in silico [29, 41, 42, 43], that utilize fitness functions based on one or some of these factors, or through the use of heuristics for special purpose encodings [44, 58]. Finding appropriate general metrics τ in oligonucleotide space and practical solutions to the corresponding restriction of DNA ENCODING is an important problem for DNA based computing. In general, even for a single good choice of quality criterion τ, the encoding problem as stated is very likely to be **NP**-complete, i.e., as difficult as the problem it is supposed to help solve, and so it would not admit general solutions. Relaxations of the problem need to be considered.

A good degree of confidence in the computation could be obtained by using a physically-based measure of error. One such measure, the *computational incoherence* (CI), has been proposed [45] based on the thermodynamics of base-stacking, DNA melting, and Gibb's free energies of duplex formation [46]. Under the assumption of equilibrium statistical mechanics, the CI estimates *the ensemble average probability of an error hybridization per hybridization event* in the test tube for a specified set of planned hybridizations; additionally, it provides an optimal reaction temperature for the experiment. Details of the derivation can be found in [45]. Laboratory experiments will help decide how good a criterion of encoding quality is captured by the CI.

4.3 Error-Preventing Codes

It is conceivable that a more principled computational approach can produce solutions of the encoding problem that capture physico-chemical conditions that are good enough to be validated by lab experiments. Perhaps the best example is the combinatorial approach proposed by the authors' molecular computing group in Memphis. The crux of the approach is to regard an experiment for a molecular computation as the transmission of a message from the protocol to the experimentalist through a noisy channel, namely the tube(s) in which the reactions take place. The theory of communication introduced by Shannon has found effective ways to handle this problem by introducing redundancy to protect against noise. The solutions are the so-called error-correcting codes for data transmission that information theorists have spent the last 50 years designing. The mathematical framework is the metric space of Boolean hypercubes with the standard binary Hamming metric. In the case of information encodings in biomolecules, one can easily generalize the Hamming distance to the four-letter alphabet `A`, `C`, `G`, `T` using Watson-Crick complementarity. This generalized Hamming metric gives some quantification of the hybridization likelihood of the molecules in the reaction. This possibility has been explored in several papers, e.g. [35, 39, 37, 47]. The problem is that oligos at a large Hamming distance can still hybridize perfectly at the overlap after a shift, such as in the case of the two strands in $\begin{array}{l}\texttt{agatcTGC}\\ \texttt{TACtctag}\end{array}$. The physico-chemical reality of the tube makes it clear that the Hamming distance is not an adequate measure of hybridization likelihood, except in very special circumstances.

Nonetheless, frame shifts appear to be accountable for, at the expense of technical complications in the Hamming metric, by a generalization, the so-called h-metric, introduced by Garzon et al. [48]. This metric may capture enough of the reality of reaction conditions and the complexity of test tube hybridizations, to frame and solve the encoding problem appropriately. The h-measure is defined as the minimum of all Hamming distances obtained by successively shifting and lining up the WC-complement of y against x; the h-metric is defined for so-called poligos, namely equivalence classes of $n-$mers at an h-measure 0 of each other. (The h-measure is not, strictly speaking, a metric.) If some shift of one strand perfectly matches a segment of the other, the measure is reduced to the value of the Hamming distance between the shifted strands. Thus a small measure indicates that the two oligos are likely to stick to each other one way or another; a large measure indicates that *under whatever physico-chemical conditions* y finds itself in the proximity of x, they are far from containing WC complementary segments, and are therefore less likely to hybridize, i.e., they are more likely to avoid an error (unwanted hybridization). Therefore the h-metric can be regarded as measuring the degree of inertia against hybridization. A solution of the encoding problem would thus consist of finding good error-correcting codes in the higher-dimensional DNA cubes. They furnish encodings that are, by definition, capable of detecting or correcting errors in the protocols by, better yet, *preventing* them from occurring, independently of the type of experiment they are used for. Error-*correcting* approaches have been suggested [49, 30, 31] but they attempt to correct errors that have been allowed to occur. Instead, it appears reasonable to *avoid* as many errors as possible, and perhaps use error-correcting methods to handle errors that cannot be prevented with the encodings suggested by, for example, the h-metric. On the other hand, most of the codes in the analogous theory of error-correcting codes are obtained by cyclic permutations of certain words, and are therefore of very low quality for the h-distance. The search has to begin anew. Genetic searches using the h-metric as a fitness function have turned up encodings that have proven to have good error-preventing properties in the lab. This is encouraging evidence that they are worth further efforts to devise algorithms to produce them, at least for $n-$mers in the range $n \leq 20$, where most experiments are currently performed.

Zhang and Shin [50] have adopted a more algorithmic approach to search for good encodings by building an evolutionary algorithm that counts the number of mishybridizations as a fitness function. They further propose that programming molecular computers should be done through artificial evolution, as is the practice in genetic algorithms for example. A mutagenesis approach to solving the encoding problem has been pursued by the MIT group [36, 42] which shows how rare good encodings may be and how hard it may be to find them. Of about 9 billion possible encodings for the design of a counter to be implemented in DNA molecules, only a handful of them survived the filtering required by various tests on the quality of the encodings derived from thermodynamical factors and lab experiments. Finally, Garzon et al. [43] have proposed a more wholistic fitness function that is based on long-term outcomes of a simulation in a virtual test tube. The simulation takes place in a cellular automaton-like environment that includes: (a) environmental factors such as temperature, salinity, etc.; (b) soft nucleotides held together by virtual covalent and hybridization bonds; and, (c) *localized* (soft) molecular interactions regulated by current knowledge of the thermodynamics of hybridization and ligation, in addition to spatial and temporal environmental constraints.

4.4 Building and Programming Molecular Computers

For several reasons, the greatest engineering and technological challenge posed by molecular computing is perhaps the construction of a molecular computer. In a molecular computer one would expect to find the basic features that are evident in a conventional electronic computer in an integrated system, namely information storage, programmability, and information processing. Such features are obviously desirable, but whether they are actually realizable is not very clear. Early papers have suggested abstract architectures (a notable example is the sticker architecture of Roweis et al. [51]) that did not focus on the issues of reliability discussed earlier. It is now clear that such issues present the most important difficulties. The best effort to date is being conducted by the Wisconsin's surface computing research group at The University of Wisconsin-Madison The instruction set consists of three primitive operations *mark*, *unmark*, and *destroy*. Successful implementation of these operations would permit, in principle, to build a general-purpose molecular computer. More details can be found in [53, 21, 54, 56, 57, 58] and references therein.

Given the difficulties with implementing traditional algorithms in DNA and their potential for evolutionary-style computation, DNA computers apparently follow Michael Conrad's trade-off principle [59, 60]: "a computing system cannot at the same time have high programmability, high computational efficiency, and high evolutionary adaptability." He describes programmability as the ability to communicate programs to the computing system exactly with a finite alphabet in a finite number of steps. The efficiency of a computing system is defined as the ratio of the number of interactions in the system that are used for computing and the total number of interactions possible in the system, and the evolutionary adaptability is defined as the ability of the system to change in response to uncertainty. It is clear that biomolecules offer, by the nature of their function, a good answer to adaptability. If Conrad's principle holds here, there is good evidence that molecular programming will be a great challenge.

4.5 Implementing Evolutionary Computation

Evolutionary computation is based on analogies of biological processes implemented in electronics to arrive at computer programs that sometimes outperform software designed by standard methodologies. The most common analogy used is natural selection, or survival of the fittest. The various methodologies include genetic algorithms, genetic programming, evolution strategies, evolutionary programming, and immune systems. These algorithms use a generate-and-evaluate strategy: a population of possible solutions is maintained (usually generated at random); individuals are then selected from a population based upon their fitness, i.e., how well they satisfy an external constraint; the population is then updated by replacing less-fit individuals by combinations of hopefully fitter individuals through some variation operations such as crossover and mutation. The basic evolution program (EP) algorithm is shown in Fig. 4. Through successive generations, the fitness of individuals is improved and better solutions are found that may converge to a good enough solution. The key ingredients in an evolutionary algorithm are selection pressure (provided by the fitness function) and *variation pressure* (provided by the genetic operations). Variation guarantees a fairly thorough opportunity for each solution to access the population of solutions and thereby a chance to be evaluated; selection guarantees that evaluation does produce better and better solutions.

```
Begin
    While Termination Condition Not True
    Begin
        Generate New Population
        Evaluate New Population
        Alter New Population with Crossover and Mutation
    End
End
```

Figure 4: Basic Algorithm for an Evolutionary Program.

A major problem faced by evolutionary algorithms is the strain they place on computational resources and running times. Beowulf clusters (large clusters of PCs) have difficulty supplying the computational power required. Molecular computing offers a great challenge but also great potential for the implementation of evolutionary algorithms. The massive parallelism of molecular computing offers an alternative that not only alleviates the computational demand for evolutionary algorithms but also takes advantage of the hybridization errors that were so troublesome in the encoding problem. The idea was proposed as a solution to the encoding problem in [39]. Beginning with a random encoding represented as circular double strands, the naturally occuring thermodynamics in the tube can be used as a fitness function to provide selection pressure, i.e., the better the desired hybridizations, the better the encodings. The selection pressure can be implemented in the tube by a hobbled repair mechanism, which is found in cells [11]. For example, the enzyme, *uvrABC*, detects mismatches in double-stranded DNA, and removes 12 base pairs (bp) from one of the strands surrounding the mismatch [11]. Further adding polymerase will rewrite the encoding in the vicinity of the original mismatch to a new, perfectly matched one. Another way to implement selection is to eliminate less fit encodings by adding, instead of polymerase, a combination of exonucleases that destroys looped molecules with mismatches, and so double-stranded molecules without loops at the end. (This is the reason for selecting circular molecules: The individuals in the population of encoding need to be protected with loops at both ends in order to prevent them from being digested by the exonucleases.)

Over several iterations of this step, however, the evaluation and selection should result in a very homogeneous population of perfectly hybridized double-stranded encodings. Some variation pressure is required to ensure a wide search of the encoding space. Controlling the concentration of enzyme and the reaction time at each step may help but is probably not enough. The equivalent of a mutation effect can be achieved by a mutagenesis technique that turns the undesirable hybridization errors into advantages. The equivalent of a crossover effect can be achieved by inserting in the encodings a blunt restriction site (for example, `AGCT` of *aluI*) and adding the appropriate concentration of the enzyme. Chen et al. [40] suggest a preliminary experiment in this direction. More details of the experiment are reported in [61].

The implementation of evolutionary algorithms in biomolecules presents an atractive alternative to further evolutionary computation research by pushing the analogy into full-fledged implementation in natural bioware and bringing the field about full circle. Molecular computing is thus poised to enable feasible solutions of hitherto infeasible search problems by using newly available molecular biological technology.

A newly emerging class of biologically inspired systems is based on the immune system. An immune system is capable of combating a large number of different types of invading pathogenic micro-organisms [11]. To accomplish this, the molecular agents of the immune system, T-cells, B-cells, and antibodies, recognize foreign antigens by structural and chemical properties of the binding sites between them. In order to do so, the immune system must be able to distinguish cells and molecules that belong to its host from foreign material, or self from nonself. Also, the immune system has a memory since it will respond to a specific antigen for the remainder of the host individual's life [11]. Over 10^{16} antigens can be recognized by a mammalian immune system.

An artificial immune system based on molecules has been suggested in [62] to duplicate the ability of a natural immune system to recognize self from nonself in order to protect a computer system from computer viruses and other unwanted agents. For discrimination of self from nonself in a computer, the entities of interest are not molecules or microorganisms, but are strings composed from a finite alphabet. These strings can be bit strings, data strings, or strings of machine instructions. For computer security, self is defined as strings to be protected, and nonself as all other strings. The steps are:

1. Detector Set Generation: Strings are generated at random. They are compared to the set

of self strings. If a matching condition between the strings is met, then, reject the string. Otherwise, accept the string for the detector repertoire. This step is called censoring.

2. Monitor Protected Data: The protected strings are periodically compared to the detector repertoire. When detector strings are activated, a change is known to have occurred.

A biomolecular implementation of such a system has been proposed in [62] based on hybridization. For the censoring, a random set of $n-$mers is generated. A self set is then constructed from the Watson-Crick complements of the encoded $n-$mers. Many copies (a picomole $\approx 10^{12}$) of the random set and self set of oligos are mixed together at high temperature. The temperature is lowered so that hybridization takes place. At this point, the self $n-$mers will have hybridized with their Watson-Crick complements in the random set. Since the self set was composed of the Watson-Crick complements of the self strings, the random $n-$mers that have hybridized correspond to the self strings. Now an enzyme, *exonuclease III*, is added to the tube. This enzyme chops up the double-stranded hybridization products into mononucleotides, effectively removing the self strands from the mix. Due to the usual errors, not all copies of the self strands will have been removed. The process of adding the self set, hybridizing, and adding exonuclease would have to be repeated to effectively remove all the self strands. At the end of the process, the remaining oligonucleotides represent the detector set. These oligos are then sequenced by gel electrophoresis or by using a DNA chip and sequencing by hybridization. More discussion on this procedure can be found in [62].

It is clear from the role that biomolecules play in natural systems and the results in this section that molecular and evolutionary computing have a good deal to gain from each other. The interaction has been fostered by two workshops [14, 63] that explored subjects such a nucleic acid selection and in-vitro evolution, topics of clear relevance to both evolutionary computation and biology.

4.6 Autonomy and Self-Assembly

It is folk knowledge now that human intervention is a bottleneck in molecular computing, i.e., it will be necessary to automate molecular protocols as much as possible. These are usually referred to as “single-pot” protocols, after Winfree [24]. These concerns have been addressed in one way or another in several works, particularly Winfree’s self-assembling reactions for tilings [64, 24, 65, 66], fault tolerance in error-preventing codes and self-control of nondeterminism and molecule formation and reaction efficiency. Jonoska and Karl [67] show how many computations can be simplified by constructing appropriate graphs in DNA molecules. Hagiya [68] has further iterated the importance of self-controlled and autonomous protocols that would eliminate human intervention and so reveal more about the true power of molecular computing. Garzon et al. [69] provide a self-assembly protocol for a family of graphs, the Cayley graphs of so-called *automatic groups*, that exploits the symmetry of the graphs and good encodings to make self-assembly possible by the type of thermocyclying effective in whiplash PCR computations.

Given the increasing importance of reliability for molecular programming, self-assembly and self-regulation are important tools to achieve a solution to the autonomy problem of molecular computers.

4.7 Molecular Computability and Complexity

Every computational paradigm eventually generates its own notion of complexity because, as pointed out early on by Charles Babbage, there is always the problem of minimizing the time and resources necesary to carry out its computational procedures. The two fundamental questions are: What problems can be solved *in principle* by molecular computing? What problems can be solved *effectively*?

The first question was addressed very early in molecular computing. Naturally, in comparing with the established standards, simulations of Turing machines were suggested that could be implemented in DNA molecules if one could only make use of arbitrarily long molecules and execute error-free operations. Bennet [70] used imaginary enzymes to show how to simulate the transitions of arbitrary Turing machines. Wilhem and Rothemund [71] show the same with commercially available enzymes. Smith [72] starts with a similar result, but further provides a somewhat negative assessment of the feasibility of DNA based computing. On the other hand, DNA capabilities have also been explored for problems on the fringe of what is theoretically solvable on VLSI computers by Lipton [17] (all NP problems) and others [74] (all **PSPACE** problems, those solvable in polynomial space by Turing machines).

Likewise, complexity measures have largely followed the standards of Boolean and formal complexity theory, i.e., using the input size used for conventional algorithm analysis and selecting a resource that the protocols spend as they are executed (see Garey-Johnson [2] for detailed definitions). The resource has usually been the steps to be executed in carrying out the protocols in a biochemical laboratory. A particularly interesting early result is Winfree's characterization of the power of three-way branched DNA to generate the derivation trees proper of context-free languages [73], in contrast with Head [13]'s early prediction that enzymes and double-stranded DNA could only generate regular languages. The complexities are usually linear or quadratic regardless of the algorithmic complexity of the original problems, so the usefulness of this analysis is not very clear. An initial attempt to characterize complexity of DNA-based algorithms in terms of the traditional concepts of "time" and "space" is introduced in [75]. These approaches essentially amount to measuring the amount of human effort involved, as opposed to measuring the resources consumed by the procedures themselves.

It is increasingly clear, however, that understanding the actual power of biomolecules to solve computational problems in practice requires developing a notion of complexity that captures the physico-chemical reality in which they take place (entirely different from VLSI-based programs), so that its results can be used to gauge the true computational power of molecular computing. Molecules are complex structures possessing physico-chemical properties that cannot be entirely described by a syntactic string giving their composition, in isolation from environmental conditions, such as concentrations (as noted by Kurtz et al. [76]). A new approach of this sort has been proposed by Garzon et al. [77]. They argue that molecules are assembled in a *tube* Ψ, which can be abstractly described as a multiset of molecules having a number of physical attributes. There are, in particular, four important properties of a tube Ψ: *volume* $V(\Psi)$, *temperature* $T(\Psi)$, *number of nucleotides* $n(\Psi)$ and *amount* $N(\Psi)$ (picomoles) of each kind of molecule contained in Ψ. (In this notation, mention of Ψ will be generally omitted.)

Therefore algorithm analysis in DNA computing should be tackled with tools that bear direct relevance to the number of molecules floating in a solution of a given volume and density in a small tube, sensitive to temperature variations, and subject to operations of various degrees of complexity and implementation difficulty (and therefore more or less expensive depending on the operation). A tool of this sort for algorithm analysis will allow comparing procedures that solve the same problem in order to determine their relative efficiency more objectively than profiling on isolated runs of the experiment, or even comparing algorithms for different problems on a common yardstick, and eventually finding lower bounds on their molecular difficulty.

Garzon et al. [77] obtain numerical complexity values that allows comparing the quality and/or difficulty of protocols for the same problem and even for different problems. They conclude that molecular computing has been making steady progress, doubling its efficiency and speed roughly every two years.

5 Conclusions

Important events have taken place in the field of biomolecular computing in the last five years. Adleman's paper landmarked a new area that has come of age thanks to the great advances in molecular biology and biotechnology of the last two decades. The initial burst of enthusiasm produced a good number of protocols and potential applications that made us realistically entertain, for the first time in history, the possibility of exploiting the massive parallelism and nanoscales inherent in natural phenomena for computational purposes.

In the process, practicioners have also come to realize that the unbridled use of molecules can quickly offset the potential gain offered by these advantages by introducing errors that render the protocols infeasible or unreliable. There indeed remain enormous scientific, engineering, and technological challenges to bring this paradigm to full fruition, i.e. make biomolecular computing a competitive player in the landscape of practical computing. Whether molecular computers will really happen in the near future will depend not only on whether these challenges can be met, but, perhaps more importantly, on whether molecular computing successfully carves a niche of "killer" applications that would continue to energize research efforts in the field.

Acknowledgements

Useful suggestions by three anonymous referees are gratefully acknowledged. Our appreciation extends as well to the editors, Moshe Sipper and David Fogel, for careful reading and comments that improved the presentation of this survey.

APPENDIX: A Resource List

There are several frequently updated web pages that contain more references and useful links to molecular computing. They include:
`http://www.msci.memphis.edu/~garzonm/mcg.html`,
`http://dope.caltech.edu/DNAevents.html`,
`http://seemanlab4.chem.nyu.edu/`, and
`http:"http://www.wi.LeidenUniv.nl/ jdassen/dna.html`.
In addition, here is a summary of events in the field since 1995. They can be accessed from the first web page above.

DNA Conferences and Meetings
DNA6 (LCNC-Netherlands, 2000)
DNA5 (MIT, 1999)
On-going Workshop in Leiden (Summer 1998)
DNA3, DNA4 (UPenn, 1997-1998)
DNA1, DNA2 (Princeton, 1995-1996)

Special program at Genetic Programming conferences:
GECCO-99 (Orlando, 1999)

GP-3 (UW-Madison, 1998)
GP-2 (Stanford, 1997)
DIMACS Workshop on Evolution as Computation [63]:
`http://dimacs.rutgers.edu/Workshops/Evolution/`;
DIMACS Workshop on Nucleic Acids Selection [14]:
`http://www.princeton.edu/~lfl/poster.html/`;
Special Session on DNA Based Computing at the IEEE-ICEC'97 conference;
Skövde meeting (1997);
Paun's meeting in Rumania (1997).

Conference Proceedings

Proceedings of the DIMACS Workshops on DNA-based Computing 1999 (MIT), 2000 (Leiden Center for Natural Computing);
1997-1998 (Upenn) [21, 54],
1995-1996 (Princeton) [52, 53],
GECCO-99 (GP-99, Orlando, 1998)
GP-98 (UW-Madison, 1998)
GP-97 (Stanford, 1997)

Surveys

DNA and Molecular Computing and Programming: this survey.
Computing with Biomolecules: theory and Experiment [74]
Fundamenta Informaticae [38]
David H. Wood's notes [80]

Conference Reports

Ferreti-Paun's' on Leiden's Workshop-98:
`http://www.dsi.unimi.it/ ~ferretti/rep.htm`;
Amos-Kari's on Leiden's Workshop-98:
`http://www.csc.liv.uk/~martyn/pubs.html`;
Amenyo's on DNA1-2-3:
`http://ftp.ans.net/pub/jta/DNAComp3rept.txt/`;
Baker's on DNA1:
`http://www.baker.com/if/dna-computer.html`.

Software

MIT's BIND, SCAN, and CYBERCYCLER [78, 36, 79].
Memphis's virtual test tube EDNA [43].

References

[1] L. Adleman, "Molecular computation of solutions of combinatorial problems," *Science* **266**, pp. 1021-1024, 1994.

[2] M.R. Garey, D.S. Johnson, *Computers and Intractability.* New York: Freeman, 1979.

[3] L. Landweber, L. Kari, "The Evolution of Cellular Computing: Nature's Solution to a Computational Problem," in [54], pp. 3-13.

[4] A. Cukras, D. Faulhammer, R. Lipton, L. Landweber, "Chess games: A model for RNA-based computation," in [54], pp. 15-26.

[5] D. Faulhammer, A. Cukras, R. Lipton, L. Landweber, "When the Knight Falls: On Constructing an RNA Computer," in [55], pp. 1-7.

[6] R. Birge, "Protein-based Three-Dimensional Memory," *The American Scientist* **82**, pp. 348-355, 1994.

[7] R. Birge, "Protein-based Computers," *Scientific American*, 1995.

[8] A.D. Ellington, M.P. Robertson, K.D. James, J.C. Fox, "Strategies for DNA Computing," in [21], pp. 173-184.

[9] M.P. Robertson, J. Hesselberth, J.C. Fox, A.D. Ellington, "Designing and Selecting Components for Nucleic Acid Computers," in [55], pp. 183-188.

[10] E.T. Kool, "New Molecular Strategies for Joining, Pairing, and Amplifying," in

[11] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, 4th ed., *Molecular Biology of the Gene*. Menlo Park, CA: The Benjamin/Cummings Publishing Co., Inc, 1987.

[12] F.M. Ausubel, R. Brent, R.E. Kingston, D.D. Moore, J.G. Seidman, J.A. Smith, K. Struhl, P. Wang-Iverson and S.G. Bonitz. *Current Protocols in Molecular Biology*, New York: Greene Publishing Associates and Wiley-Interscience, 1993.

[13] T. Head, "Formal language theory and DNA: An analysis of the generative capacity of specific recombination behaviors," *Bull. Math. Biology*, pp. 49-73, 1985.

[14] L. Landweber, R. Lipton, R. Dorit, A. Ellington (organizers), *Dimacs Workshop on Nuclei Acid Selection and Computing*, Princeton University, March 1998.
`http://www.princeton.edu/~lfl/poster.html/`;
`dimacs.rutgers.edu/Workshops/NucleicAcid/index.html`.

[15] Q. Ouyang, P.D. Kaplan, S. Liu, A. Libchaber, "DNA Solution of the Maximal Clique Problem," *Science* **278**, pp. 446-449, 1997.

[16] M. Arita, A. Suyama, M. Hagiya, "A Heuristic Approach for Hamiltonian Path Problems with Molecules," in [81], pp. 457-462, 1997.

[17] R. Lipton, "Using DNA to solve NP-complete problems," *Science* **265**, pp. 542-545, 1995. See also, "Speeding up computations via molecular biology," [52], pp. 67-74.

[18] M. Ogihara, A. Ray, "DNA-Based Self-Propagating Algorithm for Solving Bounded Fan-in Boolean Circuits," in [56], pp. 725-730.

[19] M. Amos, P.E. Dunne, A. Gibbons, "DNA Simulation of Boolean Circuits," in [56], pp. 679-683.

[20] N. Morimoto, M. Arita, A. Suyama, "Solid phase DNA solution to the Hamiltonian Path Problem," in [53].

[21] H. Rubin. D. Wood (eds.), *Proc. of the Third DIMACS Workshop on DNA-Based Computers*, The University of Pennsylvania, 1997. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, vol. **48**, 1999.

[22] M. Hagiya, M. Arita, D. Kiga, K. Sakamoto, S. Yokohama, "Towards parallel evaluation and learning of Boolean μ-formulas with molecules," in [81], pp. 105-115.

[23] K. Sakamoto, D. Kiga, K. Komiya, H. Gouzu, S. Yokohama, S. Ikeda, H. Sugiyama, M. Hagiya, "State Transitions by Molecules," in [56], pp. 87-99, 1998.

[24] E. Winfree, "Whiplash PCR for $O(1)$ Computing," in [54], pp. 175-188.

[25] M. Garzon, Y. Gao, J.A. Rose, R.C. Murphy, D. Deaton, D.R. Franceschetti, S.E. Stevens Jr. . "In-Vitro Implementation of Finite-State Machines," *Proc. 2nd Workshop on Implementing Automata WIA-97.* Lecture Notes in Computer Science **1436**, Berlin: Springer-Verlag, pp. 56-74, 1998.

[26] M. Nelson, E. Raschke, M. McClelland, "Effect of site-specific methylation on restriction endonucleases and DNA modification methyltranferases," *Nucleic Acids Research*, **21**:13, pp. 3139, 1993.

[27] E. Shapiro, "A Mechanical Turing Machine: Blueprint for a Biomolecular Computer," in [55], pp. 229-230.

[28] L. Landweber, R. Lipton, M.O. Rabin. "DNA^{2}DNA Computation: A potential "Killer-App?," in [21], pp. 162-172.

[29] R. Deaton, R.C. Murphy, M. Garzon, D.R. Franceschetti, S.E. Stevens, Jr., "Good Encodings for DNA-based Solutions to Combinatorial Problems," in [53], pp. 159-171, 1995.

[30] D. Boneh, C. Dunworth, R.J. Lipton, J. Sgall, "Making DNA Computers Error-resistant," in [53], pp. 163-171.

[31] R. Karp, C. Kenyon, O. Waarts, "Error-resilient DNA Computation," *Proc. 7th Annual Symposium on Discrete Algorithms* SODA, pp. 458-467, 1996.

[32] E. Baum, "DNA sequences useful for computation," in [53], pp. 122-127.

[33] J.G. Wetmur, "Physical Chemistry of Nucleic Acid Hybridization," in [21], pp. 1-23.

[34] J. SantaLucia, Jr., H. T. Allawi, and P. A. Seneviratne, "Improved nearest-neighbor parameters for predicting DNA duplex stability," *Biochemistry*, vol. **35**, pp. 3555–3562, 1996.

[35] R. Deaton, M. Garzon, R.C. Murphy, J.A. Rose, D.R. Franceschetti, S.E. Stevens, Jr., "On the Reliability and Efficiency of a DNA-Based Computation," *Physical Review Letters* **80**:2, pp. 417-420, 1998.

[36] A. J. Hartemink, D. K. Gifford, "Thermodynamic Simulation of Deoxyoligonucleotide Hybridization of DNA Computation," in [21], pp. 25-37.

[37] R. Deaton, D.R. Franceschetti, M. Garzon, J.A. Rose, R.C. Murphy, S.E. Stevens, Jr.. "Information Transfer through Hybridization Reactions in DNA based Computing," in [81], pp. 463-471.

[38] R. Deaton, M. Garzon, J.A. Rose, D.R. Franceschetti, S.E. Stevens, Jr.. "DNA Computing: a Review," *Fundamenta Informaticae* **35**, pp. 231-245, 1998.

[39] R. Deaton, R. Murphy, J. Rose, M. Garzon, D. Franceschetti, S.E. Stevens Jr. "A DNA based Implementation of an Evolutionary Search for Good Encodings for DNA Computation," Proc. IEEE Conference on Evolutionary Computation, Indiana, 267-271, 1997.

[40] J. Chen, E. Antipov, B. Lemieux, W. Cedeño, D.H. Wood, "A DNA Implementation of the Max 1s Problem," in [82], in press.

[41] R. Deaton, M. Garzon, R.C. Murphy, J.A. Rose, D.R. Franceschetti, S.E. Stevens, Jr. "Genetic Search of Reliable Encodings for DNA-based Computation," Late Breaking papers a the *First Annual Genetic Programming Conference*, Stanford University, pp. 9-15, 1996.

[42] J. Khodor, D.K. Gifford, A. Hartemink, "Design and Implementation of Computational Systems Based on Programmed mutagenesis," in [54], 101-107; pp. 287-297, 1998.

[43] M. Garzon, R. Deaton, J.A. Rose, D.R. Franceschetti, "Soft Molecular Computing," in [55], pp. 89-98.

[44] A.G. Frutos, Q. Liu, A.J. Thiel,A.W. Sanner, A.E. Condon, L.M. Smith, R.M. Corn, "Demonstration of a word design strategy for DNA computing on surfaces," *Nucleic Acids Res.* **25**:23, pp. 4748-4757, 1997.

[45] J. A. Rose, R. Deaton, D. R. Franceschetti, M.H. Garzon, S. E. Stevens, Jr., "A Statistical Mechanical Treatment of Error in the Annealing Biostep of DNA Computation," in [82], in press.

[46] C. R. Cantor, P. R. Schimmel, *Biophysical Chemistry, Part III: The Behavior of Biological Macromolecules*, New York: Freeman, 1980.

[47] A. Marathe, A.E. Condon, R.M. Corn, "On Combinatorial DNA Word Design," in [55], pp. 75-88.

[48] M. Garzon, P. Neathery, R. Deaton, R.C. Murphy, D.R. Franceschetti, S.E. Stevens, Jr., "A New Metric for DNA Computing," in [81], pp. 472-478.

[49] L.M. Adleman, "On Constructing a Molecular Computer," in [52], pp. 1-21.

[50] B.T. Zhang, S.Y Shin, "Molecular Algorithms for Efficient and Reliable DNA Computing," in [56], pp. 735-742.

[51] S. Roweis, E. Winfree, R. Burgoyne, N.V. Chelyapov, N.F. Goodman, P.W. Rothemund, L.M. Adleman, "A Sticker Based Model for DNA Computation," in [53], pp. 1-29.

[52] R. Lipton, E. Baum (eds.), *DNA Based Computers. Proc. of the First DIMACS Workshop on DNA-Based Computers*, Princeton University, 1995. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, vol. **27**, 1996.

[53] L.F. Landweber, E.B. Baum (eds.), *DNA Based Computers II, Proc. of the Second DIMACS Workshop on DNA-Based Computers*, Princeton University, 1996. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, vol. **44**, 1999.

[54] H. Rubin. D. Wood (eds.), 4th DIMACS workshop on DNA Computers, University of Pennsylvania, 1998. *Proceedings* in a special issue of *Biosystems*, in press.

[55] E. Winfree, D. Gifford (eds.), *Proc. of the Fifth 5th International Metting on DNA Based Computers*, MIT, Boston, MA. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Providence, RI: American Mathematical Society, In press. `http://psrg.lcs.mit.edu/dna5/`.

[56] J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo (eds.), *Proc. 3rd Annual Genetic Programming Conference*, San Mateo, CA: Morgan Kaufmann, 1998.

[57] L.M. Smith, R.M. Corn, A.E. Condon, M.G. Lagally, A.G. Frutos, Q. Liu, A.J. Thiel. "A Surface-Based Approach to DNA Computation," *J. Comput. Biology* **5**:2, pp. 255-267, 1998.

[58] A.G. Frutos, L.M. Smith, R.M. Corn, "Enzymatic Ligation Reactions of DNA Words on Surfaces for DNA Computing," *J. Am. Chem Soc.* **120**:40, pp. 10277-10282, 1998.

[59] M. Conrad, "On Design Principles for a Molecular Computer," *Comm. of the Ass. Comp. Mach. CACM*, **28**:5 1985, pp. 464-480, 1985.

[60] M. Conrad, "Molecular and Evolutionary Computation: the Tug of War between Context-Freedom and Context-Sensitivity," in [54], pp. 117-129, 1998.

[61] J. Chen, E. Antipov, B. Lemieux, W. Cedeño, D.H. Wood, *In vitro* Selection for a Max 1s DNA Genetic Algorithm," in [55], pp. 23-37.

[62] R. Deaton, M. Garzon, J.A. Rose, "A DNA Based Artificial Immune System for Self-NonSelf Discrimination," Proc. of the IEEE Int. Conference on Systems, Man and Cybernetics, Orlando. Piscataway, NJ: IEEE Press, pp. 369-374, 1997.

[63] L. Landweber, E. Winfree, R. Lipton, S. Freeland (organizers), Workshop on Evolution as Computation, Princeton University, January 1999.
`http://dimacs.rutgers.edu/Workshops/Evolution/`.

[64] E. Winfree, "Universal computational via self-assembly of DNA: some theory and Experiments," in [53], pp. 191-213.

[65] E. Winfree, "Simulations of computing by self-assembly," in [54], pp. 213-240.

[66] E. Winfree. F. Liu, L.A. Wenzler, N.C. Seeman, "Design and Self-Assembly of Two-Dimensional DNA Crystals," *Nature* **394** 1998, pp. 539-544, 1998.

[67] N. Jonoska, S.A. Karl, "Ligation Experiments in DNA Computations," *Proceedings of 1997 IEEE International Conference on Evolutionary Computation* (ICEC'97), April 13-16, pp. 261-265, 1997.

[68] M. Hagiya, "Towards Autonomous Molecular Computers," in [56], pp. 691-699, 1998.

[69] M.H. Garzon, R.J. Deaton, Ken Barnes 1999. "On Self-Assembling Graphs in Vitro," in [82], in press.

[70] C.H. Bennet, "The Thermodynamics of Computation—a Review," *Int. Journal of Theoretical Physics* **21**, pp. 905-940, 1982.

[71] P. Wilhem, K. Rothemund, "A DNA and restriction enzyme implementation of Turing Machines," in: [53] pp. 75-119, 1996.

[72] W. Smith, "DNA Computers in vitro and vivo," in: [52] pp. 121-185, 1996.

[73] E. Winfree, "On the computational power of DNA annealing and ligation," in [52], pp. 199-215, 1995.

[74] G. Paun (editor), *Computing with Biomolecules: theory and Experiments.* Singapore: Springer-Verlag, 1998.

[75] M. Amos, A. Gibbons, P. Dunne. "The Complexity and Viability of DNA Computations," *Proc. Biocomputing and Computation* (BCEC97), Lundh, Olsson and Narayanan (eds.), Singapore: World Scientific, 1997.

[76] S. A. Kurtz, S. R. Mahaney, J. S. Royer, and J. Simon, "Active transport in biological computing," in [53], pp. 111-122.

[77] M, Garzon, N. Jonoska, S. Karl. "The Bounded Complexity of DNA Computing," in [54], in press.

[78] J. Khodor, D. Gifford, "The Efficency of the Sequence-Specific Separation of DNA Mixtures for Biological Computation," in [21], pp. 25-37.

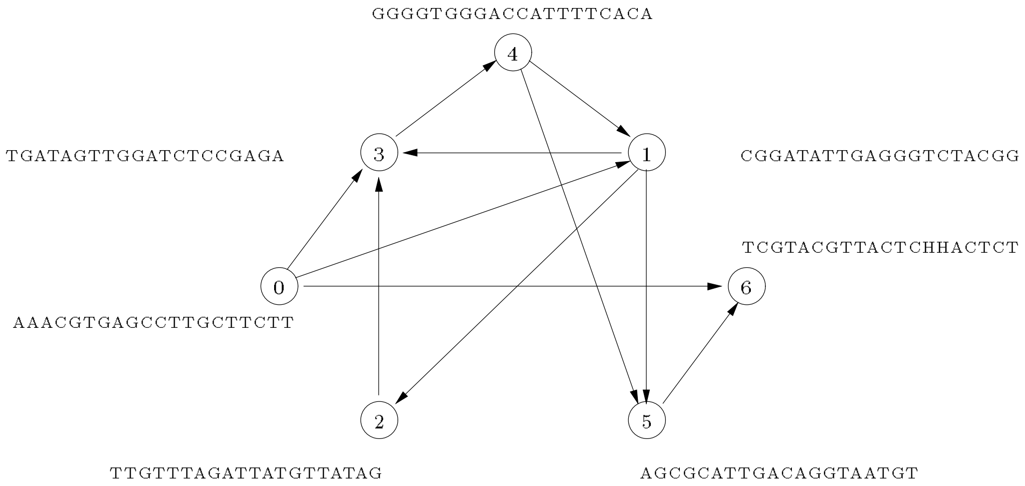
[79] A.J. Hartemink, T. Mikkelsen, D. K. Gifford, "Simulating Biological reactions: A Modular Approach," in [55], pp. 109-119.

[80] D.H. Wood 1998, *Basic DNA Computing*, manuscript.

[81] J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo, (eds.), *Proc. 2nd Annual Genetic Programming Conference*, San Mateo, CA: Morgan Kaufmann, 1997.

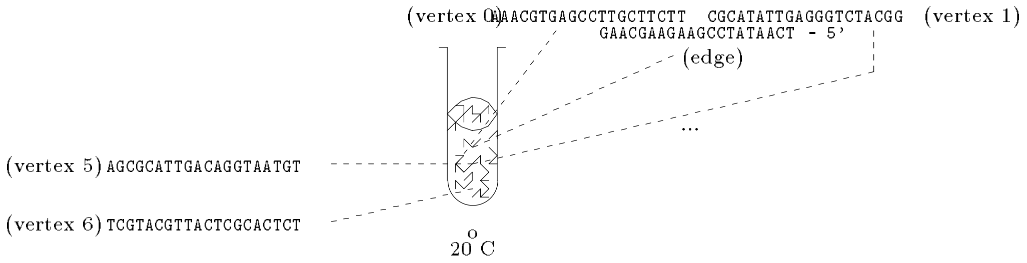
[82] W. Bahnzhaf, A.E. Eiben, M.H. Garzon, D.E. Goldberg, V. Hanovar, M. Jakiela, J.R. Koza 1999. *Proc. of the Gentic and Evolutionary Computation Conference GECCO-99*, Orlando Florida. San Mateo, CA: Morgan Kaufmann, in press.

[83] M. Chee, R. Yang, E. Hubbell, A. Berno, X. C. Huang, D. Stern, J. Winkler, D. J. Lockhart, M. S. Morris, and S. P. A. Fodor, "Acessing genetic information with high-density DNA arrays," *Science*, vol. **274**, pp. 610-614, 1996.



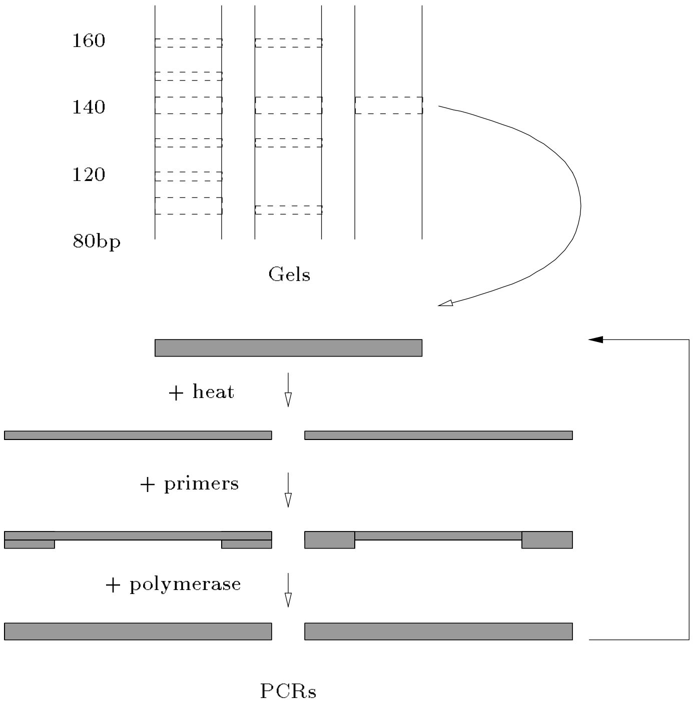
(a)

Figure 1: Steps in Adleman's Molecular Computation: (a) encoding of problem instance.


(vertex 0) AAACGTGAGCCTTGCTTCTT
CGCATATTGAGGGTCTACGG (vertex 1)
GAACGAAGAAGCCTATAACT - 5'
(edge)
...
(vertex 5) AGCGCATTGACAGGTAATGT
(vertex 6) TCGTACGTTACTCGCACTCT
20°C

(b)

Figure 1: Steps in Adleman's Molecular Computation: (b) computing reactions.



(c)

Figure 1: Steps in Adleman's Molecular Computation: (c) extraction.

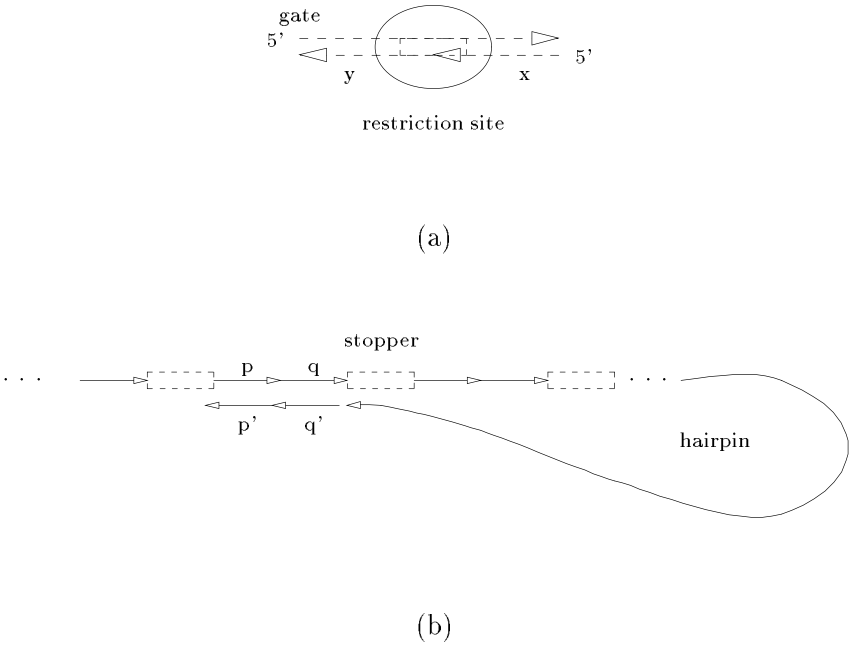


Figure 2: Two Protocols in Molecular Computation: (a) Boolean circuit implementation; (b) whiplash PCR.

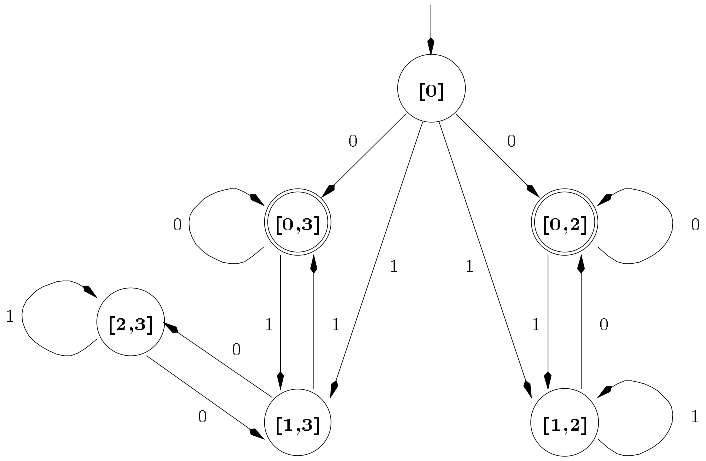


Figure 3: A Nondeterministic Finite-State Machine.

```
Begin
    While Termination Condition Not True
    Begin
        Generate New Population
        Evaluate New Population
        Alter New Population with Crossover and Mutation
    End
End
```

Figure 4: Basic Algorithm for an Evolutionary Program.