On Constructing A Molecular Computer
DRAFT*
January 8, 1995

Leonard M. Adleman†

Department of Computer Science
University of Southern California
Los Angeles, California 90089, USA

Abstract

It has recently been suggested that under some circumstances computers based on molecular interactions may be a viable alternative to computers based on electronics. Here, some practical aspects of constructing a molecular computer are considered.

1 Introduction

In [Ad] a small instance of the so called 'Hamiltonian path problem' was encoded into molecules of DNA and solved in a test tube using standard methods of molecular biology. It was asserted that for certain problems, molecular computers might compete with electronic computers. At the time that [Ad] appeared, there seemed to be formidable obstructions to creating a practical molecular computer. Roughly, these obstructions were of two types:

- Physical obstructions arising primarily from difficulties in dealing with large scale systems and in coping with errors.
- Logical obstructions concerning the versatility of molecular computers and their capacity to efficiently accommodate a wide variety of computational problems.

*This is a work in progress. Considerable modification may occur before publication.

†Research supported by the National Science Foundation under grant CCR-9214671. email: adleman@cs.usc.edu.

Happily, there has been rapid and significant progress toward overcoming these obstructions. Lipton [Li] has directly addressed the question of versatility by demonstrating a large class of computational problems which are apparently amenable to molecular computation. Further, Lipton's work has simplified the underlying theory to such an extent that it may now be possible to give a clearer picture of how to handle the physical obstructions.

This note gives one (incomplete) view of how a molecular computer might be constructed. The goal is to take a small step in the direction of practicality in the hopes that other researchers will go far beyond.

2 Abstract models

In this section abstract models of molecular computers are described. In the sections which follow physical incarnations of some of these models are given. The reader most interested in practical issues may wish to go immediately to *An Example Application:* in section 3.

We begin with the model described in [Li].

The Unrestricted (DNA) Model:

A (test) tube is a set of molecules of DNA (i.e. a multi-set of finite strings over the alphabet $\{A, C, G, T\}$). Given a tube, one can perform the following operations:

1. *Separate.*[1] Given a tube T and a string of symbols S over $\{A, C, G, T\}$, produce two tube $+(T, S)$ and $-(T, S)$, where $+(T, S)$ is all of the molecules of DNA in T which contain the consecutive subsequence S and $-(T, S)$ is all of the molecules of DNA in T which do not contain the consecutive subsequence S.
2. *Merge.* Given tubes T_1, T_2, produce $\cup(T_1, T_2)$ where:
$$\cup(T_1, T_2) = T_1 \cup T_2$$
3. *Detect.* Given a tube T, say 'yes' if T contains at least one DNA molecule, and say 'no' if it contains none.
4. *Amplify.* Given a tube T produce two tubes $T'(T)$ and $T''(T)$ such that $T = T'(T) = T''(T)$.

[1] Lipton uses the term Extract

These operations are then used to write 'programs' which receives a tube as input and return as output either 'yes' , 'no' or a set of tubes. For example, consider the following program:

(1) Input(T)

(2) $T_1 = -(T, C)$

(3) $T_2 = -(T_1, G)$

(4) $T_3 = -(T_2, T)$

(5) Output(Detect(T_3))

On input a tube, it returns 'yes' if the tube contains a DNA molecule which is composed entirely of A and otherwise returns 'no'. If step (5) is changed to OUTPUT(T_3) then it returns the tube containing exactly those DNA molecules from the input tube which are composed entirely of A.

We next describe a restricted and modified version of the unrestricted model. It is likely that some computational problems which can be efficiently handled using the unrestricted model cannot be efficiently handled using the restricted model. However, it is hoped that the restricted model will be more easily implemented.

First, despite its use in [Ad], the *Amplify* operation is a concern. Amplification is a complex and rare process. Currently, it can be only be applied to certain special biological molecules (e.g. DNA and RNA) and some living things (e.g. *E. coli.*, phages). It involves the construction of covalent bonds, the consumption of materials and may be prone to error. For the purposes of a 'practical' molecular computer, it may be preferable to avoid it (or restrict its use). Consequently, in what follows *Amplify* will not be used.

Second, it may be preferable to use molecules other than DNA, so we will start with an alphabet Σ which is not necessarily $\{A, C, G, T\}$. Further, though DNA has a natural structure which allows one to order the occurrence of symbols (5' to 3') and hence speak of 'sequences', this may not be true for other types of molecules where the groups encoding the symbols may simply be placed anywhere on a standard (potentially non linear) molecular backbone. Hence elements of a tube will no longer be sequences of symbols from Σ, but rather subsets of symbols from Σ. Such a subset will be called an 'aggregate' . Finally, separation will be allowed with respect to symbols only.

The Restricted Model:

A tube is a multi-set of aggregates over an alphabet Σ. Given a tube, one can perform the following operations:

1. *Separate.* Given a tube T and a symbol $s \in \Sigma$, produce two tube $+(T,s)$ and $-(T,s)$ where $+(T,s)$ is all of the aggregates of T which contain the symbol s and $-(T,s)$ is all of the aggregates of T which do not contain the symbol s.

2. *Merge.* Given tubes T_1, T_2, produce $\cup(T_1, T_2)$ where:

$$\cup(T_1, T_2) = T_1 \cup T_2$$

3. *Detect.* Given a tube T, say 'yes' if T contains at least one aggregate and say 'no' if it contains none.

Despite the restrictions, there are still problems of apparent interest which can be efficiently solved with the restricted model. For example, consider the well know '3-colorability problem'. The '3-colorability problem' is to decide of an undirected (i.e. with 'two-way' edges) graph $G =< V, E >$ whether each vertex in the graph can be colored either red, blue, or green in such a way that after coloring, no two vertices which are connected by an edge have the same color. The problem is NP-complete.

Given an n vertex graph G with edges $e_1, e_2, ..., e_z$, let

$\Sigma = \{r_1, b_1, g_1, r_2, b_2, g_2, ..., r_n, b_n, g_n\}$.

and consider the following restricted program on input:

$T = \{\alpha | \alpha \subseteq \Sigma \;\&\; \alpha = \{c_1 c_2, ..., c_n\} \;\&\; [c_i = r_i \; or \; c_i = b_i \; or \; c_i = g_i], i = 1, 2, ..., n\}$

(1) Input(T).
(2) for $k = 1$ to z:
  (a) $T_{red} = +(T, r_i)$ and $T_{blue\ or\ green} = -(T, r_i)$.
  (b) $T_{blue} = +(T_{blue\ or\ green}, b_i)$ and $T_{green} = -(T_{blue\ or\ green}, b_i)$.
  (c) $T_{red}^{good} = -(T_{red}, r_j)$.
  (d) $T_{blue}^{good} = -(T_{blue}, b_j)$.
  (e) $T_{green}^{good} = -(T_{green}, g_j)$.
  (f) $T' = \cup(T_{red}^{good}, T_{blue}^{good})$.
  (g) $T = \cup(T_{green}^{good}, T')$.
where $e_k =< i, j >$.
(3) Output(Detect(T)).

Clearly the elements of the input tube T are in one to one correspondence with the possible ways to assign colors to the vertices of G. Step (2) of the program creates from the current tube a new tube from which all aggregates α with $c_i = c_j$ (i.e. which assign the same color to vertex i and j) have been removed. After $5k$ separations, $2k$ merges and 1 detect, the program will output 'yes' if G is 3-colorable and 'no' otherwise. Hence the restricted program answers the 3-colorability problem for the graph G in 'linear' time (in the number of edges) while an electronic compute might require exponential time. This is the value of parallelism, each steps act on as many as 3^n inputs simultaneously.

Lipton [Li] gives a method for efficiently solving the well known (NP-complete) 'satisfiability problem': given a propositional formula (not necessarily in conjunctive normal form) decide if it is satisfiable. In fact, given a propositional formula ϕ in n variables and m binary connectives (and's or or's) and any number of not's, Lipton's method will after $m+1$ separations and m merges produce two tubes, the first contains molecules encoding satisfying truth assignments, the second contains molecules encoding unsatisfying truth assignments. A single final detect on the first tube determines whether ϕ is satisfiable or not.

For most of the incarnations of restricted computation which are considered here, merging is done by pouring the contents of the input tubes into a single tube. This is presumably much faster and less error prone than separating. For many of the problems attacked by restricted computers, in appears that detect is done rarely. Accordingly, when considering a proposed incarnation, the following parameters will be considered:

Time: The time τ required for a separation.

Accuracy: The error rate of a separation. In a real system separation will not be perfect, there will be errors of inclusion and exclusion. Let ϵ_+ denote the probability that after a separation, an aggregate which should end up in $+(T, s)$ actually ends up in $-(T, s)$. Let ϵ_- denote the probability that after a separation, an aggregate which should end up in $-(T, s)$ actually ends up in $+(T, s)$.

It is of course possible that in real systems the time and accuracy of a separation may depending on the tube and symbol being considered. However, for convenience it will be assumed that a uniform time and accuracy can be given.

The restricted model of molecular computation is memory-less in the sense that the molecules themselves do not change in the course of a computation. The state of the computation is represented primarily by the distribution of the molecules in the various tubes. Next an abstract model of a molecular computer with memory is presented. This model has the same power as the unrestricted model. It is presented for its potential to be realized in a practical way; however, it will be given only passing attention in what follows and can be safely omitted by the reader more interested in practical matters.

The Memory Model:

A tube is a multi-set of aggregates over an alphabet $\Sigma = \{a_1, a_2, ..., a_n, b_1, b_2, ..., b_n\}$. Given a tube, one can perform the following operations:

1. *Separate* Given a tube T and a symbol $s \in \Sigma$, produce two tubes $+(T, s)$ and $-(T, s)$ where $+(T, s)$ is all of aggregates of T which contain the symbol s and $-(T, s)$ is all of the aggregates of T which do not contain the symbol s.

2. *Merge.* Given tubes T_1, T_2, produce $\cup(T_1, T_2)$ where:

$$\cup(T_1, T_2) = T_1 \cup T_2$$

3. *Detect.* Given a tube T, say 'yes' if T contains at least one aggregate and say 'no' if it contains none.

4. *Flip.*

   (a) Given a tube T and a symbol $a_i \in \Sigma$, produce a new tube $T^{a_i} = \{\alpha^{a_i} | \alpha \in T\}$ where for all $\alpha \subset \Sigma$, if $a_i \notin \alpha$ then $\alpha^{a_i} = \alpha$ and if $a_i \in \alpha$ then $\alpha^{a_i} = (\alpha - \{a_i\}) \cup \{b_i\}$.

   (b) Given a tube T and a symbol $b_i \in \Sigma$, produce a new tube $T^{b_i} = \{\alpha^{b_i} | \alpha \in T\}$ where for all $\alpha \subset \Sigma$, if $b_i \notin \alpha$ then $\alpha^{b_i} = \alpha$ and if $b_i \in \alpha$ then $\alpha^{b_i} = (\alpha - \{b_i\}) \cup \{a_i\}$.

Hence the flip operations switches a_i to b_i (b_i to a_i) in all aggregates that contain an a_i (b_i).

Let

$$T = \{\alpha | \alpha \subseteq \Sigma \ \& \ \alpha = \{l_1, l_2, ..., l_n\} \ \& \ [l_i = a_i \ or \ l_i = b_i] \ i = 1, 2, ..., n\}$$

Then we can think of each aggregate in T as a memory with n locations, where the i^{th} location contains a 1 if $l_i = a_i$ and contains a 0 if $l_i = b_i$. We can 'execute' the following types of instructions: 'for all aggregates in the tube T, if location 3 has a 1 and location 5 has a 0, then set location 51 to 1' for example with the following molecular program:

(1) Input(T)
(2) $T_1 = +(T, a_3)$
(3) $T_2 = +(T_1, b_5)$
(4) $T_3 = T_2^{b_{51}}$
(5) $T' = \cup(-(T, a_3), -(T_1, b_5))$
(6) $T = \cup(T', T_3)$

3 The DNA computer

In this section we consider a DNA based implementation of the 'restricted model' (see section 2) of molecular computer.

We will require that the alphabet of symbols Σ contain two special symbols $s_{5'}$ and $s_{3'}$. For each symbol $s \in \Sigma$ an oligonucleotide (i.e. short sequence of nucleotides, oligo) is chosen. The resulting set of oligos should satisfy the following properties:

(1) Under easily achievable conditions (for example of temperature, pH, salt concentration, ion concentration) each oligo reliably forms stable hybrids with its Watson-Crick complement.

(2) Under easily achievable conditions each oligo reliably dissociates from its Watson-Crick complement.

(3) Under neither of the conditions above does any oligo form hybrids with itself or another oligo, nor with another oligo's Watson-Crick complement.

For the DNA computer we will require that all input tubes consist of molecules of DNA which begin (5') with the oligo associated with $s_{5'}$ and end (3') with the oligo associated with $s_{3'}$.

If a *Merge* of tubes is required, this is accomplished by pouring the contents of all tubes into a single tube.

If a *Separate* is required on a tube T with respect to a symbol s, then if s is associated with the oligo O, a separator which consists of molecules of the oligo Watson-Crick complementary to O conjugated to solid supports are used. For example, the magnetic bead system employed in [Ad], or an affinity column.

If a *Detect* is required on a tube T, then a PCR with primers appropriate for the common 5' and 3' sequences are used followed by a gel electrophoresis.

An Example Application:

To gauge what may be possible with a DNA computer, we will consider an example. Consider solving an instance of the 'satisfiability problem' [2]. Given a propositional formula ϕ with say 70 variables and 1000 binary connectives. Then using Lipton's method, $\Sigma = \{T_i | i = 1, 2, ..., 70\} \cup \{F_i | i = 1, 2, ..., 70\} \cup \{s_{5'}, s_{3'}\}$ - a 'true' symbol and a 'false' symbol for each of the variables and the required 'PCR symbols' $s_{5'}$ and $s_{3'}$. Associating each symbol of Σ with a randomly choosen (or carefully designed) 10-mer would seem likely to satisfy the three conditions listed above. Denote the oligo associated with T_i as O_i^T, the oligo associated with F_i as O_i^F and the oligos associated with $s_{5'}$ and $s_{3'}$ as $O_{5'}$ and $O_{3'}$ respectively. One would want the input tube T to consist of the set of all DNA molecules of the form:

$$O_{5'} M_1 M_2 ... M_{70} O_{3'}$$

where for $i = 1, 2, ..., 70$, $M_i = O_i^T$ or $M_i = O_i^F$. Thus the elements of T are in one to one correspondence with the set of all possible truth assignments for the 70 variables.

There would be 2^{70} molecules, each of which is a 720-mer. Assuming that each nucleotide has molecular weight 300 dalton, then the molecules in tube T have a mass of approximately 425 grams (0.93 lb.[3]). While this is a rather large quantity by the standards of current molecular biology, there do not seem to be reasons in principle why dealing with such quantities would not be feasible in practice.

To create T one could proceed as described in [Li] using the methods of [Ad] (see also [BS]). However, it is perhaps more reliable to proceed in the following manner - well known in 'combinatorial chemistry' [Al]. First, all of the O_i are produced separately. Then approximately 2^{70}, $O_{5'}$ are conjugated to solid support in a tube T. Then the following steps are taken for $i = 1, 2, ..., 70$:

(1) pour half of the contents of tube T into T_1 and half into T_2.
(2) To the molecules in T_1 ligate (a single) O_i^T to the 3' end.

[2] The 'satisfiability problem' is to distinguish satisfiable propositional formulas from those which are not satisfiable. For example '(A or B) and (not C and not A)' is a propositional formula with 3 variables (A,B,C) and three binary connectives (one 'or' and two 'and'). This propositional formula is satisfiable since the truth assignment which sets A to False, B to True, and C to False, makes the formula evaluate to True. The formula '(A and not A)' is not satisfiable since no truth assignment makes the formula evaluate to True. The 'satisfiability problem' is NP- complete and the best known algorithms for it require exponentially time.

[3] Warning: all calculations performed with a Pentium 90.

(3) To the molecules in T_2 ligate (a single) O_i^F to the 3' end.
(4) Merge T_1 and T_2 into T.

Finally to the molecules in T ligate (a single) $O_{3'}$.

This is on a scale larger than customary in molecular biology and also creates 720-mers which are longer than currently produced by standard methods. Nonetheless, these problems appear technical in nature and creating T by this or other means seems feasible.

Performing the required merges would be trivial.

Performing detection by PCR would be easily accomplished by standard means. Recall that such a PCR is only performed once at the end of the computation. If the final tube contained many DNA molecules (i.e. if there were many truth assignments which satisfied ϕ) then the detect operation could be performed with coarser techniques such as optical density measurement.

Performing separations might be done using the magnetic bead system of [Ad]. Each bead is approximately 1 micron in diameter and has a mass of approximately 1.3 pg. Each bead can bind approximately 7.8×10^5 biotinylated oligos. Assuming that one bead-associated biotinylated oligo will bind 1 target molecule, then to bind 2^{69} target molecules (the number of targets in the largest separation done during this computation) would require approximately 984g (approximately 2.2 lb.) of beads with a volume of roughly 0.39 liters (approximately 0.49 qt.)[4]. While separation by this method is on a scale unusual in molecular biology, it seems quite feasible. It is important to note that these beads are reusable. After a separations, the beads and their oligos are intact except for 'natural degradation'.

Assume that each separation could be done in 1000 sec. (i.e. τ =approximately 16.7 minutes - this seems a reasonable estimate) and that the time for a merge is negligible. Also assume that there are no errors during separations (i.e. $\epsilon_+ = \epsilon_- = 0$ - this is unreasonable, but how non-zero values affect the results will be considered separately in the following section (section 4)). Let an operation be taken as either having a molecule of DNA 'decide' to go into one tube or another during a separation, or having a molecule of DNA transferred from one tube to another during a merge. The number of separations is 1001 and the number of merges is 1000. For a typical ϕ, it would seem reasonable to assume that each separation and each merge would be applied to a tube with approximately 2^{69} DNA molecules, hence the number of operations per second

[4] Because spheres do not pack without spaces, a container for these beads would have a volume of approximately 750 ml (approximately 0.79 qt.)

would be approximately 1.2×10^{18}. This is approximately $1,200,000$ times faster than the fastest super computer. The total time for the calculation would be dominated by the 1001 separations and hence would be approximately 11.6 days. During the computation approximately 10^{24} operations would be performed, which is quite possibly greater than the total number of operations performed on man-made computing devices though out history[5].

It is also worth noting that while the molecular computer above would require 140 'separators' (a 'True' and 'False for each propositional variable), Lipton's algorithm makes use of only one at a time. Hence as many as 140 propositional formulas might be handled by the machine simultaneously. In this case, the machine might execute as many as 1.6×10^{20} operations per second.

4 Error Control

In this section we will consider the implications of errors during separation. To begin, we will take care of a technical point. Each incarnation of a restricted computer is associated with the parameters $\tau, \epsilon_+, \epsilon_-$ (see section 2). It could be the case that ϵ_+ and ϵ_- are very different. For example in the DNA computer described in section 3, the process of separation might typically be carried out in two steps:

(1) *Anneal.* Incubate the contents of T with the beads and allow time for molecules with the correct consecutive subsequence to anneal. Pour off the liquid phase containing molecules that do not anneal.

(2) *Wash.* Add solvent to the beads and pour off. This will remove additionalany molecules which do not annealed to beads (for example molecules left on the walls of the tube or held weakly to the beads). Repeat.

One can imagine that once a molecule that should be in $+(T, s)$ sticks to a bead that it will stay stuck (with very high probability) though many washings. So that ϵ_+ is determined by how well the molecules in $+(T, s)$ stick during the anneal step. On the other hand molecules which weakly anneal to the beads or are stuck to the sides of the tube may only be removed on a second or third washing. So ϵ_- may depend on how much washing is done. One can imagine a system with several washings which would give values like (these are strictly conjectural) $\epsilon_+ = 1/10$ and $\epsilon_- = 1/10^6$.

[5]We have not addressed the energy consumption aspects of molecular computers in this paper. Nonetheless, one can expect that molecular computers will use much less energy than electronic computers. Indeed, it may be the case that separations can be performed 'reversibly'.

In the analysis that follows it will be preferable to have ϵ_+ and ϵ_- approximately equal. This is always possible to achieve with the use of repeated separations:

(first) Input(T)
(1) $T_1 = +(T, s)$ and $T_1' = -(T, s)$.
(2) $T_2 = +(T_1', s)$ and $T_2' = -(T_1', s)$
.
.
.
(n) $T_n = +(T_{n-1}', s)$ and $T_n' = -(T_{n-1}', s)$
(last) $T^+ = \cup(T_1, T_2, ..., T_n)$ and $T^- = T_n'$.

Then after n steps T^+ will have $(1 - \epsilon_+^n)$ of the elements which should be in $+(T, s)$ and will have $1 - (1 - \epsilon_-)^n$ of the elements which should be in $-(T, S)$. Further, $T = \cup(T^+, T^-)$. When n is choosen so that ϵ^n approximatly equals $1 - (1 - \epsilon_-)^n$ (which itself approximately equals $1 - n\epsilon_-$), then we can think of the n repeated separations, yielding the tubes T^+ and T^- from T, as a single new separation operation with ϵ_+ approximately equal to ϵ_- and τ n times as large as the original τ. For example, if the original separation had $\epsilon_+ = 1/10$, $\epsilon_- = 1/10^6$ and $\tau = 1000$ sec., then the new separation (with $n = 6$ - actually $n = 5.28$ is better but we will use an integral number of repetitions) has $\epsilon_+ = 1/10^6$, ϵ_- approximately $6/10^6$ and $\tau = 6000$ sec.

So hence forth we will assume that $\epsilon_- = \epsilon_+$ and we will denote this value as ϵ. We will assume that for our satisfiability example $\epsilon = 6/10^6$.

Now assume that a given molecule enters s separations in the course of a computation (e.g. in the satisfiability example $s \leq 1000$). What is the probability p_{good} that this molecule goes through all separations without a mistake (i.e. without ever going in to the 'wrong' tube). That probability is:

$$(1 - \epsilon)^s$$

and the probability p_{bad} that it fails to go through the s separations without a mistake is at most:

$$1 - (1 - \epsilon)^s$$

For the satisfiability example, for all molecules $p_{good} \geq (1 - 6/10^6)^1000$ which is approximately 0.994 and $p_{bad} \leq 0.006$. Hence at least 994 molecules out of each 1000 make it through the entire calculation with no error. This is pretty good, but not adequate, because $\frac{6}{1000}2^{70}$ (approximately 1.5×2^{62}) molecules could have errors at some point and might end up in the wrong tube when the computation is done. To handle this problem consider the following:

Assume you have a traditional sequential electronic computer. There is a predicate P (i.e. a function which on each input returns either 'yes' or 'no') and assume you write a program which exhaustively (and sequentially) searches all 2^{70} binary strings of length 70 for the existence of a 'winner' string for which P gives a 'yes' answer. Further assume that for each string of length 70, P can be computed with 1000 operations. If during the search, the computer stops and prints out 'α is a winner', should you believe it? Like molecular computers, electronic computers are physical systems which are subject to error. If you are told that your computer makes at most one error in $2^{1,000,000}$ operations, then, as a practical matter, you are likely to be justified in believing that α is indeed a winner. If however, you are told that your computer makes at most one error every $166,666$ operations, then the situation is not so clear. You may want to check the computer's answer by retesting the string α several times. Each time you get the answer 'α is a winner' (without ever getting the answer 'α is a loser'), you are justified in increasing your belief that α is indeed a winner - but of course you will never be absolutely sure.

These ideas will now be applied to the molecular setting. Assume that p_{good} is fairly large (as in the satisfiability example) and that we have run the computation for the propositional formula ϕ and have produced the output tube T_1^w which is supposed to (i.e. if there were no errors) have the 'winners' (i.e. those molecules which encode satisfying truth assignments for ϕ). What if we now take the tube T_1^w and use it as the input to an entire rerun of the whole computation (i.e. retest all of the purported 'winners')? We will then get a new tube T_2^w which is all of the molecules which went through the entire computation twice and both times ended in the winning tube. What if we repeat that process through r runs of the entire computation - what now will be in T_r^w? It will contain two types of molecules: the 'true winners' (those that really do encode satisfying truth assignments) and the 'false winners' (those which encode unsatisfying truth assignments - but have somehow, through errors, made it into T_r^w anyway). Notice that it is much easier to get into T_r^w as a 'true winner' than as a 'false winner'. A 'true winner' need only go through r computations without an error - but most molecules do make it though each pass of the computation without an error (in our example 994 of every 1000 at least). In fact, the probability of making it through r passes is at least p_{good}^r. On the other hand, a 'false winner' is unlikely to make it through to T_r^w. A 'false winner' must have at least one error on each pass through the computation (otherwise on the pass without errors, it would end where it belongs - out of the winner tube). But only at most p_{bad} (in our example 6 in 1000) make an error in each pass. To make an error r times in a row has a probability of at most p_{bad}^r. In our example we could choose $r = 10$. Then p_{bad}^{10} is approximately $1/(13.5 * 2^{70})$. And hence the expected number of 'false winners' in tube T_{10}^{w} is at most $1/13.5$ (less than one). Hence even if ϕ is not satisfiable, it is unlikely some 'false winner' will end up in T_{10}^{w} and make us believe that it is satisfiable. Now p_{good}^{10} is approximately 0.94. So even if there is just one satisfying truth assignment for ϕ, there is at least a 94% chance the molecule encoding that assignment will end up in T_{10}^{w} - and hence we will be likely to give the correct answer, that ϕ is satisfiable.

It is important to note that the analysis above was rather heavy handed and that a more refined analysis would probably give a better value for r. In addition, there may well be better ways to handle separation errors than those proposed above. It seems unlikely that the approach described is optimal either mathematically or physically. On the physical side there is probably much room for improvement. The sorts of separations actually needed are quite special. First, the oligos used can be specifically choosen to diminish the possibility of non-specific binding. Second, the tubes are free of extraneous molecules (e.g. enzymes) which might complicate the interactions. Third, there is a symmetry in this setting. For example, if a separation with respect to the symbol F_5 is done then $-(T, F_5) = +(T, T_5)$, since every molecule in the system has either T_5 or F_5 but not both. Consequently one could imagine carrying out the separation using a 'double positive' system. Consider putting the contents of the tube into a chamber which contains beads for T_5 on the left and beads for F_5 on the right with a semipermeable membrane in between which allows oligos to pass but not beads (indeed with such a system, the beads might be replaced by large molecules which cannot pass the membrane). Agitation of the chamber might yield very clean separations - experimentation would allow one to be sure. Such a system might also be very fast and perform separations without the need to expand the volume of solvent used. At any rate, even the analysis done here gives reason to be encouraged that separation error control will not be an onerous burden.

There are of course other possible sources of error in molecular computation; for example, 'natural degeneration' of molecules. In a DNA computer, it would of course be necessary that major sources of degeneration such as nuclease contamination be avoided and it would be sensible to have separations done under the mildest conditions possible. It might also be wise to use analogues of DNA such as those with peptide backbones which might be more robust. In any case, it seems unlikely that a reasonable system for controlling such errors could not be designed.

5 Other Incarnations: The Inorganic Catalytic Molecular Computer

In this section the prospects for an improved incarnation of a molecular computer are considered.

Because the amplify instruction is not used in the restricted model, there may no longer be a need to use biomolecules for computation. Using inorganic molecules may have several advantages.

First, these molecules could potentially be much smaller than those used in the DNA computer. In the DNA computer each symbol is associated with a 10-mer which has a molecular weight of approximately 3000 dalton. In an inorganic computer one might hope to associate each symbol with a much smaller molecule. Since roughly speaking mass is inversely proportional to the amount of parallelism available, such small molecules could lead to restricted molecular computers which much greater parallelism than the DNA computer.

Second organic molecules might be constructed which were extremely stable. This could result in a 'catalytic' molecular computer. Since merge and separate do not involve the making or breaking of covalent bonds, if we assume that detect can be done without the construction or destruction of molecules, then once a computation is completed, all resulting tubes need only be merged into a single tube to recreate the input. Hence the molecular computer in a real sense catalyzes the computation. This may be important as a practical matter. For example, one could build a molecular computer for satisfiability which would create the input tube T (consisting of the molecules encoding all possible truth assignments) just once. Then given a formula ϕ, T would be used to determine whether ϕ was satisfiable. Once that computation was completed, the tube T would be recreated and the next formula processed[6].

Third, given the freedom to choose arbitrary molecules, it may also be possible that a set of inorganic molecules might be choosen with properties that allow for systems with small values of τ, ϵ_+, and ϵ_-.

What an appropriate set of molecules would look like is best left to professional chemists. However, to provide an unsophisticated starting point for discussion, consider the following system. Associate each symbol with a small functional group (e.g. methyl, amino). Form an aggregate by bonding the appropriate functional groups to a fullerene skeleton. For the purposes of separation, use

[6] By implementing *Detect* carefully on a DNA computer, it can be made 'catalytic' or nearly so.

'combinatorial' chemistry or rational design to develop enzymes which specifically bind to each incorporated functional group. Such a system, if possible, would encode an aggregate in approximately one fiftieth of the mass used by the DNA computer. Hence it would provide approximately 50 times the parallelism. Further fullerenes are apparently quite stable.

The memory model might (in theory) be implemented by a system like the following. For each 'location' i, a_i is associated with an oligonucleotide O_i and b_i is associated with a methylated version of O_i. Enzymes are used to methylate and demethylate. For example, if $O_i = 5'...GAATCC...3'$ then EcoR I methylase will catalyze the transfer of a methyl group (from S- adenosylmethionine) resulting in $5...GA^mATCC...3'$. EcoR I methylase will not methylate oligos without the subsequence $GAATCC$. Unfortunately, specific enzymes to demethylate appear not to be known.

An inorganic incarnation, might also be possible. As above, each a_i is associated with some small functional group and b_i is obtained from a_i by the non-covalent attachment of a small molecule which binds to a_i.

Building a computer with a small τ (i.e. time of separation, see section 2) would be very desirable. One of the problems with the designs described to this point is that 'separation' is a largely mechanical process applied from outside the tubes. It would be interesting to design a molecular computer (perhaps based on entirely different 'primitives' than those used here) which would accomplish its task by purely chemical means inside of a single tube.

Since the restricted model of computation requires only merges, separations and detections, there may be other physical systems which could provide an incarnation. For example, some form of all of these operations appear possible using atomic or subatomic particles traveling in accelerators with separation done according to mass or charge.

6 Detect

In going from an unrestricted computer to a restricted one, the amplify operation was removed. Nonetheless in the DNA computer, amplification was still used in order to implement the detect operation. If a restricted computer based on some other material is to be built, then some means of performing detect without amplification may be required. There appear to be reasons to be optimistic about the existence of such systems. It may be possible to detect single molecules in solution by labeling them with fluorescent ligands [ER]. It appears that physicists are able to detect a single subatomic event in a huge volume of space when detecting neutrinos.

Another operation which might be considered in a molecular computer is *Describe*. Roughly, decode a molecule back into the set of symbols it encodes. For example, when determining whether a propositional formula ϕ is satisfiable or not, a molecular computer may produce a tube which contains the molecules that encode satisfying truth assignments (if such exist). One may wish to have an explicit description of one such truth assignment (rather than to just know that at least one exists). Hence one would want to take a molecule from the tube and *Describe* it. This operation would be implementable in the DNA computer by well know methods (e.g. sequencing) of molecular biology. It is worth noting however that it is in fact efficiently implementable in all incarnations of restricted molecular computers. Let the alphabet of the system be $\{s_1, ..., s_z\}$, then to *Describe* a single molecule which is contained in a tube T, one can use the following program:

(1) Input(T)
(2) Symbols= $\emptyset$ (begin with the empty set of symbols)
(3) If Detect(T) $= 0$ then output Symbol.
(4) For $i = 1$ to z:
  (a) If Detect($+(T, s_i)$) $= 1$ then:
    (i) $T = +(T, s_i)$
    (ii) Symbol=Symbol$\cup\{s_i\}$ (add an s_i to the set of symbols)
  (b) If Detect($+(T, s_i)$) $\neq 1$ then:
    (i) $T = -(T, s_i)$
(5) Output Symbol

Notice that even if T contained many different molecules each encoding a different set of symbols, the program above would give an explicit description of exactly one of them.

7 Applications to Biology, Chemistry and Medicine

It is the author's hope that the ideas used in designing and programming molecular computers will be of direct (non-computational) value in biology, chemistry and medicine. The ability to create molecules with desired properties (e.g. enzymes, drugs) 'on demand' is of great importance in these areas. Historically, the process of creating such molecules has been difficult (if possible at all) and expensive. Very recently, a new promising approach to this problem, called 'combinatorial' chemistry [Al], has arisen. Below is an example of its use. It should be clear that combinatorial chemistry is closely related to molecular computation. It seems possible that concepts useful in designing molecular computers may be of value in combinatorial chemistry. Following the example a potential application of these concepts is described.

Recently, Bartel and Szostak [BS] have used the methods of combinatorial chemistry to make what might be called a 'pseudo-enzyme'. We will describe their basic approach briefly. For the sake of clarity many simplifications will be made and many important details will be omitted.

The goal of the experiment was to find a molecule of RNA which would ligate two substrate molecules of RNA ρ_1 and ρ_2 (ρ_1 and ρ_2 were base pair complementary in such a way that hybridization would bring their 5'-triphosphate and 3'-hydroxyl groups into proximity in preparation for ligation). A pool of approximately 4^{25} random sequences of RNA was developed. Each RNA in the pool was bound to a copy of ρ_2 at the 5' end and a copy of a constant region C at the 3' end. Hence, a tube containing approximately 4^{25} molecules was created, and each molecule in the tube had the form: $\rho_2 RC$ where R was a some molecule of RNA from the original pool. To this tube an excess of ρ_1 was added. If a 'winner' W existed in the RNA pool which could ligate ρ_1 to ρ_2 then in the tube $\rho_1\rho_2 WC$ would be formed. Next molecules which contained the sequence ρ_1 were separated from the rest and retained (hence 'non winner' RNA from the original pool were removed). Because the 5' sequence of ρ_1 was known and the sequence C was known, it was now possible to amplify (using reverse transcip-tion, PCR, transcription) only those molecules of the form $\rho_1\rho_2 WC$. Standard means could now be used to discover the sequences of the 'winners'.

The 'winner' (actually 'winners') of Bartel and Szostak experiment is not a true enzyme, since it is 'used up' in acting on its substrates. The value of having the 'winner' RNA 'anchored' to one of the substrates is that once the winner acts, it becomes permanently anchored to the reaction product. This distinguished it from the rest of the RNA pool and allows it to be retrieved for later identification. If one wished to create a true enzyme, one might have to identify the 'winner' despite the fact that after acting on its substrate it reenters the pool. This is an example of the problem which we will now explore further: finding the 'winner' when all that is available is the ability to know that a 'winner' exists somewhere in a large pool. Being able to solve this problem in a general setting would be particularly importanct when the pool molecule were not nucleotides (e.g. a pool of proteins, a pool consisting of multiple variants of an existing compound). Or when anchoring was not physically possible or appropriate (for example when an endonuclease was sought or a drug which crosses a cell membrane and then binds a particular host molecule). Below we show that this problem can be solved in a wide variety of settings. We demonstrate the technique with an example. Our goal will be to find an RNA molecule which will ligate (as a true enzyme) two DNA molecules δ_1 and δ_2 (which as above, hybridize would bring their 5'-triphosphate and 3'-hydroxyl groups into proximity in preparation for ligation).

We will need a little notation. Let P denote the pool of all 25-mers of RNA (there would be a total of 4^{25}) . If we have an RNA sequence σ let P_σ be all 25-mer RNA which begin (5') with σ. For example P_A would be all 25-mers which begin with A (there would be a total of 4^{24}), and P_{ACCU} would be all 25-mers which begin with $ACCU$ (there would be a total of 4^{21}).

Consider being given two 50 nucleotide sequences of DNA, δ_1 and δ_2 and seeking an RNA which will ligate them. One can begin as in [BS] by creating an RNA pool consisting of all 25-mers (i.e. begin with P). One then incubates P, and excess δ_1 and δ_2 in an appropriate buffer. After an appropriate time, one runs a PCR with primers derived from the 5' end of δ_1 and the 3' end of δ_2. Thus neither δ_1 nor δ_2 will be amplified but the product of their ligation would be. Hence if no PCR product is produced we will declare that the pool P contains no 'winner' and discontinue the experiment for that pool. If a PCR product is produced we will declare that the pool P contained a 'winner'[7] How do we find the winner? We proceed as follows. In the next step we repeat the experiment above 4 times, once with each of P_A, P_C, P_G and P_U. Since $P = P_A \cup P_C \cup P_G \cup P_U$, at least one (and perhaps more than one) of them will contain a 'winner'. Lets say P_C contains a 'winner', then we know some 'winner' begins with C. Next we repeat the experiment with P_{CA}, P_{CC}, P_{CG} and P_{CU}. Again at least one must contain a 'winner'. Say P_{CA}, then we know some winner begins with CA. Continuing in this fashion we will come to know the exact sequence of some 'winner' as desired.[8]

Admittedly this is a considerable amount of repetitious work. It requires 101 steps of pool synthesis, incubation and PCR (many of these steps could be run in parallel of course). However, in these 101 steps, 4^{25} (over 1000 trillion) RNA strings are searched. This is of course the same sort of trade off achieved for our molecular computers applied to problems like satisfiability (linear time versus exponential time). Further, by doing the incubation under 'unamiable' conditions where only the 'best' winners will have a chance to act, the need for 'evolving' the answer as done for example in [BS] is removed and this should save considerable time. Also, there is never a need to perform amplification, which in the case of RNA is laborious and for other molecules may be impossible.

It is worth remarking that this technique would work in the same way if the pool was composed of RNAs consisting of all length 25 sequences of elements

[7] For the purpose of this illustration, we will ignore certain 'practical' problems such as the occasional uncatalyzed ligation of δ_1 and δ_2.

[8] The complexity theorist will recognize the similarity of this technique to that used for demonstrating that if there exists a polynomial time algorithm for deciding satisfiability, then there exists a polynomial time algorithm for finding satisfying truth assignments.

from any set of four 'basic' RNA units (for example four 10-mers rather than the 1-mers used in the example). The RNA in the pool could also have fixed constant regions if that was desired.

It should be clear that similar techniques would work in a wide variety of settings. Using pools made of RNA, DNA or some other form of aggregates. Whether a ligation or some other function was desirable - so long as the existence of a 'winner' could be detected.

So the method demonstrates that the ability to detect the existence of a winner leads efficiently to the identification of a winner.

8 Discussion

First a general caveat. To correctly gauge the practicality of molecular computation will require inputs from experts in a wide variety of fields including: biology, chemistry, computer science, engineering, mathematics and physics. The author is not well versed in all of these areas, yet has choosen, either explicitly or implicitly, to make assumptions about each in writing this paper. Errors and omissions should be expected. Ideally, those with expertise in these areas will find ways to improve the approach or reveal fundamental flaws.

Can a practical molecular computer be built? It is still too early to know. However, the DNA computer described here gives reason for optimism.

The author suspects that molecular computers fill a computational niche which electronic computers do not and visa versa. Is it better to have one computer which can execute a trillion instructions per second, or **three** computers which can each execute **half** a trillion instructions per second? The answer depends of what problem you wish to solve. If the problem can be split into three tasks each requiring about the same number of operations to complete, then the three computers will solve the problem more quickly. If however the task is not decomposable in this way, then the one faster computer is best. Molecular computing takes this example to an extreme, instead of three computers executing half a trillion instructions per second, you are offered 10^{20} (or more) computers each of which executes one instruction every 1000 seconds or so. Which is best? If your problem can be decomposable into a huge number of tasks and each task is very simple (as in our example of satisfiability) then the molecular computer is much faster. If such a decomposition is not possible then the single faster computer is best. It is an interesting challenge to envision a system somewhere in between, which provides a moderate number of computers each operating at a moderate speed.

9 Acknowledgments

The author would like to thank Jonathan DeMarrais, David Jefferson and Ron Rivest for their comments.

References

[Ad] Adleman L. Molecular computation of solutions to combinatorial problems. *Science* 266:1021-1024 (Nov. 11) 1994.

[Li] Lipton R. Speeding up computations via molecular biology. Draft. Dec. 9, 1994. (/ftp/pub/people/rjl/bio.ps on ftp.cs.princeton.edu).

[BS] Bartel D. and Szostak J. Isolation of new ribozymes from a large pool of random sequences. *Science* 261:1411-1418 (Sept. 10) 1991.

[Al] Alper J. Drug discovery on the assembly line (Research News).*Science* 264:1399- 1401 (Jun. 3) 1994.

[ER] Eigen M and Rigler R. Sorting single molecules - applications to diagnostic and evolutionary biotechnology. *PNAS* 91: 5740-5747 (Jun. 21) 1994.