A Sticker Based Model for DNA Computation

Sam Roweis[1]*, Erik Winfree[1],
Richard Burgoyne[2], Nickolas V. Chelyapov[3],
Myron F. Goodman[4], Paul W. K. Rothemund[3],
Leonard M. Adleman[3]†

Laboratory for Molecular Science, University of Southern California
and
[1] Computation and Neural Systems Option, California Institute of Technology
[2] Department of Biomedical Engineering, University of Southern California
[3] Department of Computer Science, University of Southern California
[4] Department of Biological Sciences, University of Southern California

May 1996

Abstract

We introduce a new model of molecular computation that we call the *sticker model.* Like many previous proposals it makes use of DNA strands as the physical substrate in which information is represented and of separation by hybridization as a central mechanism. However, unlike previous models, the stickers model has a random access memory that requires no strand extension, uses no enzymes, and (at least in theory) its materials are reusable.

The paper describes computation under the stickers model and discusses possible means for physically implementing each operation. We go on to propose a specific machine architecture for implementing the stickers model as a microprocessor-controlled parallel robotic workstation. Finally, we discuss several methods for achieving acceptable overall error rates for a computation using basic operations that are error prone.

In the course of this development a number of previous general concerns about molecular computation [Smith, Hartmanis, Letters to Science] are addressed. First, it is clear that general-purpose algorithms can be implemented by DNA-based computers, potentially solving a wide class of search problems. Second, we find that there are challenging problems, for which only modest volumes of DNA should suffice. Third, we demonstrate that the formation and breaking of covalent bonds is not intrinsic to DNA-based computation. This means that costly and short-lived materials such as enzymes are not necessary, nor are energetically costly processes such as PCR. Fourth, we show that a single essential biotechnology, sequence-specific separation, suffices for constructing a general-purpose molecular computer. Fifth, we illustrate that separation errors can theoretically be reduced to tolerable levels by invoking a trade-off between time, space, and error rates at the level of algorithm design; we also outline several specific ways in which this can be done and present numerical calculations of their performance.

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

---

*Reprint requests to *roweis@cns.caltech.edu*. The MATLAB code which was used to generate all of the figures in Section 5 of this paper is also available by request from *roweis@cns.caltech.edu*. Roweis is supported in part by the Center for Neuromorphic Systems Engineering as a part of the National Science Foundation Engineering Research Center Program under grant EEC-9402726 and by the Natural Sciences and Engineering Research Council of Canada. Winfree is supported in part by National Institute for Mental Health (NIMH) Training Grant # 5 T32 MH 19138-06; also by General Motors' Technology Research Partnerships program. Adleman, Chelyapov, and Rothemund are supported in part by grants from the National Science Foundation (CCR-9403662) and Sloan Foundation.

†To whom correspondence should be addressed.

1 Introduction

Much of the recent interest in molecular computation has been fueled by the hope that it might some day provide the means for constructing a massively parallel computational platform capable of attacking problems which have been resistant to solution with conventional architectures. Model architectures have been proposed which suggest that DNA based computers may be flexible enough to tackle a wide range of problems [Adleman1, Adleman2, Amos, Lipton, Boneh2, Beaver, Rothemund], although fundamental issues such as the volumetric scale of materials and fidelity of various laboratory procedures remain largely unanswered.

In this paper we introduce a new model of molecular computation that we call the *sticker model*. Like many previous proposals it makes use of DNA strands as the physical substrate in which information is represented and of separation by hybridization as a central mechanism. However, unlike previous models, the stickers model has a random access memory that requires no strand extension, uses no enzymes, and (at least in theory) its materials are reusable.

The paper begins by introducing a new way of representing information in DNA, followed by an abstract description of the basic operations possible under this representation. Possible means for physically implementing each operation are discussed. We go on to propose a specific machine architecture for implementing the stickers model as a microprocessor-controlled parallel robotic workstation, employing only technologies which exist today. Finally, we discuss methods for achieving acceptable error rates from imperfect separation units.

2 The Stickers Model

2.1 Representation of Information

The stickers model employs two basic groups of single stranded DNA molecules in its representation of a bit string. Consider a *memory strand* N bases in length subdivided into K non-overlapping regions each M bases long (thus $N \geq MK$). Each region is identified with exactly one bit position (or equivalently one boolean variable) during the course of the computation. We also design K different *sticker strands* or simply *stickers*. Each sticker is M bases long and is complementary to one and only one of the K memory regions. If a sticker is annealed to its matching region on a given memory strand then the bit corresponding that particular region is *on* for that strand. If no sticker is annealed to a region then that region's bit is *off*. Figure 1 illustrates this representation scheme.

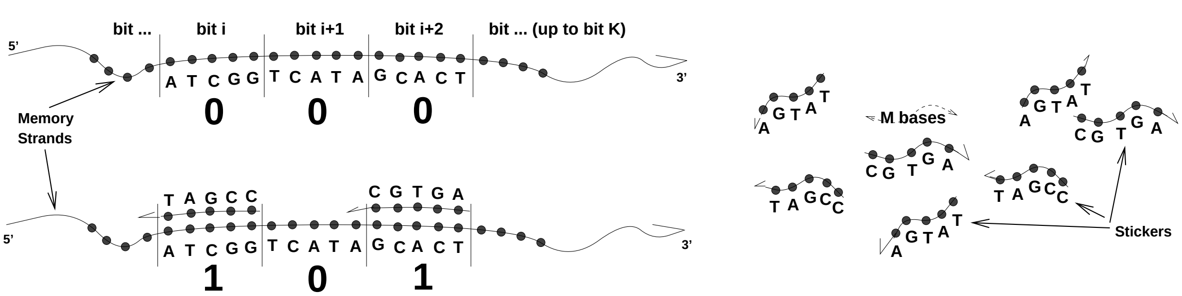

Figure 1: A memory strand and associated stickers (together called a *memory complex*) represent a bit string. The top complex on the left has all three bits *off*; the bottom complex has two annealed stickers and thus two bits *on*.

Each memory strand *along with* its annealed stickers (if any) represents one bit string. Such partial duplexes are called *memory complexes.* A large set of bit strings is represented by a large number of *identical* memory strands each of which has stickers annealed only at the required bit positions. We call such a collection of memory complexes a *tube.* This differs from previous representations of information using DNA in which the presence or absence of a particular subsequence in a strand corresponded to a particular bit being on or off (e.g. see [Adleman1, Lipton]). In this new model, each possible bit string is represented by a unique *association* of memory strands and stickers whereas previously each bit string was represented by a unique molecule.

To give a feel for the numbers involved, a reasonable size problem (for example breaking DES as discussed in [Adleman3]), might use memory strands of roughly 12000 bases (N) which represent 580 binary variables (K) using 20 base regions (M).

The information density in this storage scheme is $(1/M)$ bits/base, directly comparable to the density of previous schemes [Adleman1, Boneh2, Lipton]. We remark that while information storage in DNA has a theoretical maximum value of 2 bits/base, exploiting such high values in a separation based molecular computer would require the ability to reliably separate strands using only single base mismatches. Instead we choose to sacrifice information density in order to make the experimental difficulties less severe.

2.2 Operations on Sets of Strings

We now introduce several possible operations on sets of bit strings which together turn out to be quite flexible for implementing general algorithms. The four principle operations are *combination* of two sets of strings into one new set, *separation* of one set of strings into two new sets and *setting* or *clearing the* k^{th} *bit* of every string in a set. Each of these logical set operations has a corresponding interpretation in terms of the DNA representation introduced above. Figure 2 summarizes these required DNA interactions.

- The most basic operation is to *combine* two sets of bit strings into one. This produces a new set containing the multi-set union of all the strings in the two input sets. In DNA, this corresponds to producing a new tube containing all the memory complexes (with their annealed stickers undisturbed) from both input tubes.
- A set of strings may be *separated* into two new sets, one containing all the original strings having a particular bit *on* and the other all those with the bit *off.* This corresponds to isolating from the set's tube exactly those complexes with a sticker annealed to the given bit's region. The original input set (tube) is destroyed.
- To *set* (turn on) a particular bit in every string of a set, the sticker for that bit is annealed to the appropriate region on every complex in the set's tube (or left in place if already annealed).
- Finally, to *clear* (turn off) a bit in every string of a set, the sticker for that bit must be removed (if present) from every memory complex in the set's tube.

Computations in this model consist of a sequence of combination, separation, and bit setting/clearing operations. This sequence must begin with some initial set of bit strings and must ultimately produce one (possibly null) set of strings deemed to be "the answers". We call the tube containing the initial set of bit strings the *mother tube* for a computation. Thus, to complete our theoretical description of

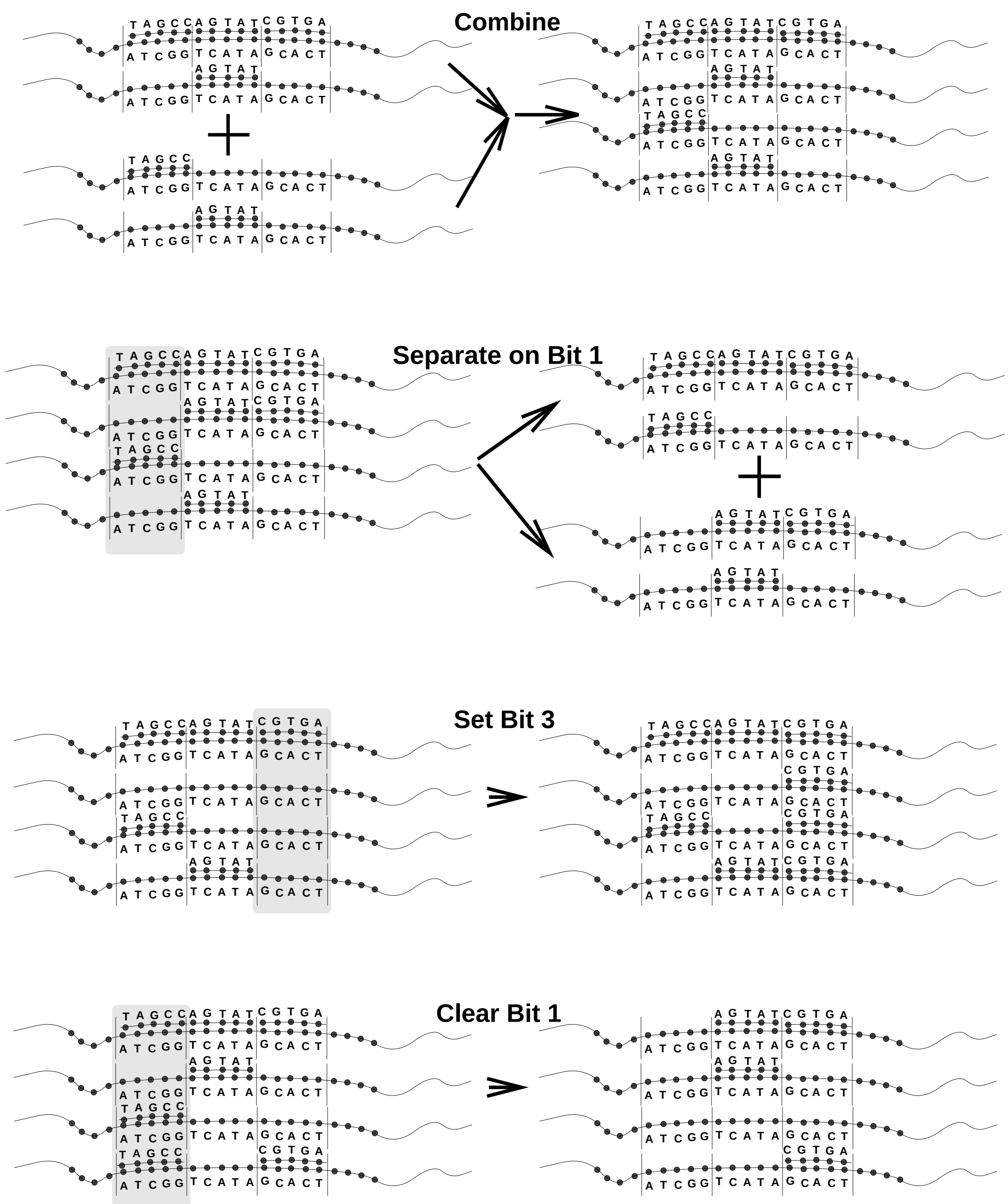

Figure 2: DNA manipulations required for the four operations of the stickers model.

how to compute with the stickers model, we must describe how to create a mother tube of memory complexes and also how to read out at least one bit string from a (possibly empty) final tube of answers (or recognize that the tube contains no strands). We consider creation of the mother tube first:

- It will suffice for our purposes to consider creating a mother tube which corresponds to the (K, L) *library set* of strings. A (K, L) *library set* contains strings of length K generated by taking the set of all possible bit strings of length L followed by $K - L$ zeros. There are thus 2^L length K strings in the set[1].

Our paradigm of computation will generally be to cast hard problems as large combinatorial searches over inputs of length L. We search for the few rare "answer" strings by processing all 2^L possible inputs in parallel and eliminating those that fail the search criteria. It is important that the memory strand we design may have more than L bit regions. The first L bits represent the encoding of the input and are the random portion of the initial library. The remaining $K - L$ bits are used for intermediate storage and answer encoding and are initially off on all complexes. All bits can be written to and read from later in the computation as needed. In this way creating a mother tube which is a (K, L) *library set* corresponds to generating all possible inputs (of length L) and zeroing the workspace (length $K - L$).

Lastly, we indicate how to obtain a solution at the end of the computation:

- To read a string from the final "answer" set, one memory complex must be isolated from the answer tube and its annealed stickers (if any) determined. Alternately, it must be reported that the answer tube contains no strands.

2.3 Example Problem

To illustrate the power of the operations defined above we work through the solution of the NP-Complete[2] *Minimal Set Cover* problem [Garey] within the stickers model. Informally, assume we are given a collection of B bags each containing some objects. The objects come in A types. The problem is to find the smallest subset of the bags which between them contain at least one object of every type. Formally the problem is as follows: *Given a collection $C = \{C_1, \ldots, C_B\}$ of subsets of $\{1, \ldots, A\}$ what is the smallest subset I of $\{1, \ldots, B\}$ such that $\bigcup_{i \in I} C_i = \{1, \ldots, A\}$?* The solution of the problem in our model is straightforward. We create memory complexes representing all possible 2^B choices of bags. We mark all those which include bag i as containing every type appearing in the subset C_i. Then we separate out those complexes which have been marked as containing all A types and read out the one(s) which uses the fewest bags. Formally, the sticker algorithm for minimal set cover is:

- `Design a memory strand with` $K = B + A$ `bit regions.` — *Bits $1 \ldots B$ represent which bags are chosen, bits $B + 1 \ldots B + A$ which object types are present.*
- `Initialize a` (K, B) `library set in a tube called` T_0.
- `for i=1 to B`
    - `Separate` T_0 `into` T_{on} `and` T_{off} `based on bit` i — *Mark the final A positions of each complex to record which object types it contains.*
    - `for j=1 to` $|C_i|$
        - `Set bit` $N + C_i[j]$ `in` T_{on}
    - `Combine` T_{on} `and` T_{off} `into` T_0

[1] For example, the (7,3) *library set* is the set {0000000,0010000,0100000,0110000,1000000,1010000,1100000,1110000}.

[2] Technically the NP-Complete version of this problem is the binary decision version in which we ask if there exists a collection of a *particular* size that covers the set, not for the collection of the smallest size.

```
• for i=B+1 to B+A                                          Get rid of ones which do not have
      Separate T_0 into T_0 and T_bad based on bit i          all A types.
      Discard T_bad
• for i=0 to B-1                                              Count how many bags
      for j=i down to 0                                       were used. At the end
          Separate T_j into T_(j+1)' and T_j based on bit i+1 of the outer loop, tube
          Combine T_{j+1} and T_(j+1)' into T_{j+1}           T_i contains all com-
•                                     Read T_1;               plexes which used ex-
  else if it was empty then           Read T_2;               actly i bags.
  else if it was empty then           Read T_3;
                                      ...
```

where above $|C_i|$ is the number of items in subset C_i and $C_i[j]$ is the j^{th} item in subset C_i. Note that the above algorithm takes $O(AB)$ steps, and the input is $O(AB)$ bits.

We point out that, as we will envision a robotic system performing the experiments automatically, we allow arbitrary sequential algorithms for controlling the molecular operations. However, these operations must be performed "blind"; the only interface to molecular parallelism is via *initialize, combine, separate, set, clear,* and *read.* Thus the electronic algorithms are responsible for "experiment design" i.e. compiling higher-level problem specifications into concise sequences of molecular operations but they cannot get any feedback from the DNA during the course of the experiment.

As a final comment we note that the stickers model is capable of simulating (in parallel) independent universal machines, one per memory complex, under the usual theoretical assumption of an unbounded number of sticker regions[3]. It should be noted that the stickers model is universal, in the sense discussed, even in the absence of the *clear* operation, although more compact algorithms are possible using *clear.*

3 Physical Implementation of the Model

Each logical operation in our model has a corresponding interpretation (which we gave as we introduced the operations) in terms of what must happen to the DNA memory strands and associated stickers when that operation is carried out. In what follows we examine various physical procedures which are candidates for implementing these requirements for all the operations described above. We speak in terms of *tubes* instead of *sets*; recall that a *tube* consists of the collection of memory complexes that represents a *set* of bit strings.

Often there are several possible implementations of a given operation; each has its own assumed strengths and weaknesses on which we speculate. However, which implementations, if any, turn out to be viable will ultimately have to be decided by laboratory experiments.

3.1 Combination

Combination of two tubes can be performed by rehydrating the tube contents (if not already in solution) and then combining the fluids together (by pouring or pumping for example) to form a

[3] This can be seen as the consequence of two observations. First, a memory complex in the stickers model can simulate a feedforward circuit, in the spirit of [Boneh2]. Using the *clear* operation, a clocked feedback circuit can also be simulated. Second, allowing the circuit to grow with each clock cycle, we can simulate a universal machine. The electronic algorithm is responsible for designing the new gates to fit into the circuit; each new gate will require a new bit and hence a new sticker region in the memory strand. For concreteness, a feedforward circuit C_t can be automatically designed which computes the instantaneous description of a TM at time step t from the description at $t-1$. Thus, the stickers model can simulate in parallel the execution of a TM on all 2^L length L inputs.

new tube. It should be noted that even this seemingly straightforward operation is plagued by constraints: if DNA is not handled gently the shear forces from pouring and mixing it will fragment it into ≈ 15 kilobase sections [Kornberg].

Also of concern for this operation and indeed for all others is the amount of DNA which remains stuck to the walls of tubes, pumps, pipette tips, etc. and thus is "lost" from the computation. Even if this "lost" DNA is a minute fraction of the total (which would be unimportant to molecular biologists) it is problematic for computation because we are working with relatively few copies of each relevant molecule.

3.2 Separation

The ultimate goal of the separation operation is to physically isolate those complexes in a tube that have a sticker annealed to some position from those that do not without disturbing any annealed stickers. The mechanism of DNA hybridization will be central to any proposal. In general, separation by hybridization is is performed by bringing the solution containing the original set of memory complexes into contact with many identical single stranded *probes*. In our case, each bit position has a particular type of probe (with a unique nucleotide sequence) that is used when separation on that bit is performed. The probe sequence is designed such that probes hybridize only to the region of the memory strand corresponding to their bit and nowhere else. During separation, the original complexes with the key bit *off* will be captured on the probes while all those with the bit *on* will remain unbound in solution because the region is covered by a sticker. Next, the unbound ("on") complexes are physically isolated, for example by conjugating the probes to magnetic beads or affixing the probes to solid support and then washing. Lastly, the "off" memory complexes are recovered from the probes that bound them by elution (say by heating and washing). The result is two new tubes, one containing the memory complexes for each of the output sets of the operation.

Notice that if heating is used to achieve the final step of elution this must be done without also removing all of the stickers from the memory strands. This necessitates that the probes have a lower binding affinity for their corresponding regions than do the stickers. This might be achieved by making the probe sequences not exactly complementary to their regions on the memory strands (or merely shorter) to create a differential between the temperature of probe-strand and sticker-strand dissociation. An alternative is to use perfectly complementary sequences for both the probes and stickers but to make the stickers out of an alternate backbone material (such as PNA or DNG [Egholm1, Dempcy]) which would exhibit stronger and more specific binding to the DNA memory strand than DNA probes[4]. PNA and DNG offer the additional advantage that decreasing salt concentration causes PNA/DNA and DNG/DNA to bind more strongly while the opposite is true for DNA/DNA binding. Thus the final elution step might be achieved by washing in a zero salt solution rather than by heating. There are other possibilities for creating differential affinity between the stickers and probes[5].

3.3 Setting and Clearing

To *set* a bit in every string of a set the most obvious choice is direct annealing. An excess amount of the sticker corresponding to the bit is added to the tube containing the set's memory complexes.

[4] PNA "clamps" [Egholm2] have been shown to form $(PNA)_2/DNA$ triplexes with remarkable affinity and specificity. These clamps could also be used as stickers.

[5] For example crosslinking techniques might be used to covalently bond the stickers to the memory strands so that they could not come off during elution, although this confounds the *clear* operation and does not keep with the reusable spirit of the model.

One sticker should anneal to every complex that does not already have one, always in the position opposite the region corresponding to the bit being set. Subsequently the excess (unused) stickers are removed, perhaps by filtration or by *separating* out all the memory complexes. This latter proposal could be achieved by having a *universal region* on every memory strand (say at the very beginning or end) that is never covered by a sticker and designing a probe for that region as described in the separation operation above. Such a universal region is a generally useful idea for recovering all memory complexes from a given solution which may contain other species.

To *clear* a bit in every string of a set requires removing the stickers for only that bit from every complex in a tube. Simple heating will obviously not work since *all* stickers from *all* bit regions will come off. One possibility is to designate certain bit regions as *weak* regions. These regions have weak stickers which dissociate more easily from the memory strand than regular stickers. By heating to some intermediate temperature *all the weak stickers* can be made to dissociate at once, keeping all of the regular stickers in place.

In order to implement the *clear* operation in full generality, it may be possible to use the phenomenon of PNA strand invasion by triple helix formation [Nielsen]. It has been shown that under appropriate conditions, two single stranded oligos of all-pyrimidine PNA will "invade" an existing complementary DNA/DNA duplex to form a $(PNA)_2/DNA$ triple helix, displacing the pyrimidine DNA strand. This process is most efficient with PNA "clamps"[Egholm2] which contain both the Watson-Crick and Hoogsteen PNA strands in a single molecule. We suggest that if for example 21 nucleotide DNA stickers are used, then a 14 base PNA clamp could be designed which forms a triple helix with the central 7 nucleotides of the DNA sticker. By mixing PNA clamps specific to a particular bit with a tube of memory complexes, and heating, the PNA clamps should form triple helices with the targeted sticker, destabilizing and thus "prying" it off at a temperature lower than the dissociation temperature for the unaffected stickers. The specificity and reliability of this operation are not yet known experimentally; indeed the mechanism of triplex formation[Demidov] may be incompatible with the requirement that non-targeted stickers remain in place. In terms of physical implementation prospects, *clear* seems to be the most problematic of our operations. Recall, however, that it can be eliminated without significantly sacrificing the computational power of the model.

3.4 Initialization and Final Output

To make a combinatorial library containing roughly one copy of every possible bit string of length L followed by $K - L$ zeros, it is first necessary to synthesize roughly 2^L identical copies of a properly designed memory strand with $K \geq L$ regions. Stickers must then be added "randomly" to these strands in positions $1 \dots L$. One procedure that achieves this is outlined below. Note that the method requires only a single step.

The strands are split into two equal volumes. To one volume is added an excess of stickers for all bits $1..L$; this results in all bits $1..L$ being set on all strands. The unused stickers are then removed, for example by filtration or by separating on a universal region of the memory strand. The two volumes are then recombined and heated causing all stickers to dissociate. Finally the mixture is cooled again, causing the stickers to randomly anneal to the memory strands. Since each bit position has only one sticker for every two strands, the resulting memory complexes have any given bit set with probability one half (very nearly independently). Under this model, the odds that any particular bit string is *not* present in the final library is $(1 - 1/2^L)^{2^L}$ which for the L of interest is almost exactly $1/e$. In other words each string is created at least once with probability roughly 63%. This percentage can obviously be increased by synthesizing more than 2^L strands initially.

Notice that this procedure is relatively robust to errors in stoichiometry: For example, if the original strands are split into volumes whose ratio is not 1 but 1.5 then (for say $L = 56$) a randomly chosen string is created with probability 37%, still not vanishingly small[6].

To obtain an output string it is necessary to be able to *detect* the presence or absence of memory complexes in a solution. If any are present, we also need to be able to isolate at least one memory complex and then identify which stickers (if any) are annealed to it.

Detection of complexes might be accomplished by fluorescent labeling of each memory strand. Single molecule detection can then be performed by running the solution through a fine capillary tube. Such detection has already been achieved experimentally, see for example [Castro]. This technique may also be effective for isolating a single complex if the time between detection events is large enough. In addition to the capillary tube method mentioned above, other proposals (e.g. based on PCR) for complex detection are possible.

The final step of identifying annealed stickers may be possible by direct imaging – since we know the order of bit regions we could imagine just *looking* and reading off the answer string (perhaps using electron microscopy). Alternately once a complex is isolated its stickers may be eluted and poured over a detection hybridization grid [Meade] to determine which ones were present. While these possibilities are intriguing, more practical approaches based on PCR are more likely to work in the near term [Adleman3]. However, we show below that detection alone is sufficient to obtain an output string. The approach is to use binary tree decoding:

Begin with the solution containing all putative answer complexes (of which there may be none). Detect complexes in it. If there are none, then no answer has been found. If there are some then separate them based on the first bit of the answer string[7]. Detect complexes in each of the resulting solutions and retain the one which is not empty. If neither is empty then there is more than one answer and either can be retained. Repeat this separation and detection for all the bits of the answer string.

3.5 Memory Strand and Sticker Design

At several points in the above discussion it was necessary to design the sequence of the memory strand or stickers to have certain properties. In this section we summarize those requirements and explore possibilities for achieving them.

The most fundamental requirement of sequence design is to achieve sticker specificity. It is critical that the stickers only anneal to the memory strands when opposite their assigned region and not in any other position. Thus the memory strand sequence must be designed so that any region's complementary sticker is only complementary to that one region and has much reduced affinity at all other alignments along the strand. As a first approximation to this we will require a certain minimum number of base mismatches at all other alignments. Notice that this is a much stronger requirement than simply requiring each sticker to mismatch all bit regions but its own. It must mismatch every other M long window (possibly spanning two bit regions) on the strand. Mathematically, we wish to design a sequence of length N such that there exist K non-overlapping subsequences of length M each (call them "regions") with the following property: For each region, its complement has at least

[6]The expression for the probability of a random bit string being created is $1 - \sum_{k=0}^{L} \frac{\binom{L}{k}}{2^L} [1 - r^k (1+r)^{-L}]^{2^L}$ where r is the ratio of the volumes into which we split initially.

[7]The answer string which we are interested in reading out may be a substring of the entire string encoded by the memory strand in which case separation only needs to be done for those bits.

D_1 mismatches with *every* other subsequence of length M in the entire sequence. The quantity D_1 is the minimum number of mismatches needed for a sticker M bases long not to anneal.

It is also important to eliminate secondary structure in the memory strand itself. We must prevent the memory strand from annealing to itself and creating a hairpin structure, as this makes regions inaccessible for proper use in the system. Fulfilling this requirement can be loosely modeled by the combinatorial problem of designing a N long sequence such that the complement of *every* subsequence of length M has at least D_2 mismatches with every other subsequence of length M. The quantity D_2 is the minimum number of mismatches to prevent the memory strand from self-annealing.

Finally, we must design separation probes such that they stick specifically to the appropriate region and they have sufficiently lower affinity there than the stickers. This ensures that there exists a wash temperature (and salinity) for which the probes will dissociate while the stickers will remain in place. Again, as a first approximation we require that the probes have at least D_3 mismatches within their region and at least $D_4 > D_3$ mismatches everywhere else.

These criteria may seem daunting. However, there are some ways to make this task potentially easier. Notice that in general we may leave portions the memory strand unused; that is we may not identify those portions with any regions so that the product of K and M does not always equal N (but certainly still $KM \leq N$). In other words, we leave "gaps" between the bit regions on the memory strand. In order to avoid the secondary structure problem, it has been suggested that the memory strand be composed of only pyrimidines (or purines) and the stickers of only purines (or pyrimidines)[Mir]. The applied mathematics literature on "comma free codes" and on "de Bruijn sequences" (when $D = 1$) contains detailed discussions of many of the important issues (see [Neveln] and [Fredricksen] for reasonable introductions). Also, [Smith, Baum] have discussed sequence design in the context of DNA computation.

Finally, D_1 would be reduced if higher-affinity PNA or DNG stickers were used; furthermore, D_3 would possibly be reduced to zero. Other variables other than or in addition to temperature could be manipulated, such as salt concentration and chemical solvent, in order to achieve the relative affinities required for each operation. It is worth speculating about the possibility of using naturally occurring sequences (e.g. plasmids) for the memory strands because of the obvious ease of their mass production. However it remains to be seen if natural sequences can be found which meet the above restrictions.

We emphasize that the criteria outlined above are for illustration only; a more sophisticated approach would have to take into consideration the sequence-dependent thermodynamic parameters for oligonucleotide hybridization. There are several data sets available for calculating ΔH and ΔS for DNA/DNA hybridization [Santalucia, Breslauer, Petruska], and similar data could be obtained for PNA and/or DNG interactions. Allowances would also have to be made for potential bubble mismatches at incorrect sticker hybridization sites, and secondary structure due to triple helix formation must be prevented. The *clear* operation, if used, would introduce additional constraints. Although such sophisticated design approaches could suggest potentially useful memory strand, sticker, and probe sequences, correct operation will have to be tested experimentally.

Our conclusion is that although design of the memory strand and the stickers may be difficult, the design space is large; and once a strand with K regions is found, it can be used and reused in the stickers model for any problem requiring K or fewer bits of memory. Since the stickers model uses only a single type of memory strand, in contrast to the 2^K different molecules required in the representation of [Boneh2], the design process is simplified and the functionality of the strand can be tested experimentally once and for all.

3.6 Experimental Feasibility

The stickers model as presented above presents challenging requirements for strand design and experimental implementation. Several objections might be raised to the effect that it is unreasonable to expect that these requirements can be met. We attempt to briefly address some of these issues here.

Objection: No matter what methods are proposed, DNA based techniques will suffer from strands being misprocessed. What error rates would be required in order to still accomplish useful computation?

Response: For many search problems, including DES and NP-complete problems, probabilistic algorithms have practical value. Answers suggested by the molecular computer, so long as there aren't too many, can be verified electronically. To ensure that a complex carrying the solution to the problem has a 90% chance of ending up in the "answers" tube after a 1000-step computation, separation error probabilities of less than 0.01% are required. To eliminate false-positive distractors, it may be necessary to refine the "answers" tube by repeating the steps of the computation [Adleman2, Karp]. This and other related error-handling strategies are discussed in Section 5.

Objection: Purity and yield of 90% for purification of DNA are considered excellent in molecular biology. The conditions imposed for separation of memory complexes are much more challenging, since long strands may be used, stickers must not be knocked off, and both supernate and eluant are required. Yet DNA computation requires much lower error rates, both for purity and yield.

Response: Isolation of particular target DNA in complicated cDNA libraries is a routine task in molecular biology. 10^5-fold enrichment of target DNA, with 80% recovery, has been reported using, for example, triplex affinity capture [Ito]. The use of PNA probes also shows some promise: 99% purification with 50% yield using PNA 15-mers has been reported [Orum]. However, current techniques do not meet our requirements for the separation operator. We do not believe that this is due to a fundamental limit. So long as yield is extremely high (i.e. memory complexes don't get "lost"), our calculations (see Section 5) suggest that a poor separation can be improved dramatically by automated processing. Furthermore, we have the opportunity to design our own sequences that can be effectively separated, for example by ensuring that the memory strand has no secondary structure. We recognize that attaining high step yield may be a major challenge, however.

Objection: Even without trying to process them at all, stickers will be falling off their memory strands at some rate k_d. Once a sticker dissociates, it may then hybridize to and thus corrupt some other complex. During operations such as separate, when memory complexes must be melted from probes, k_d surely increases. By the time the computation is complete, the contents of the memory complexes may be completely scrambled.

Response: Suppose we would like to ensure that fewer than 0.01% of stickers fall off during the course of a 1000 hour computation. This would require a k_d of less than $0.3 \times 10^{-9}/sec$. A generic DNA 20-mer can be estimated to have the required k_d at 42°C in 1 M [Na$^+$] [Wetmur]. PNA and DNG stickers would be expected to have an even lower dissociation rate, especially at low salt. High wash temperatures may be avoided by using DNA probes and PNA or DNG stickers, and washing in low salt. Additionally, we must be careful not to encourage other circumstances, such as rough physical handling, which might induce sticker dissociation.

Objection: If DNA is subjected to high temperatures for a significant portion of a 1000 hour computation, it may be damaged by deamination, depurination, or strand breakage by hydrolysis, thus rendering it non-functional. (Such objections are discussed briefly in for example [Smith].)

Response: Under physiological conditions of salinity, pH, and temperature, the depurination half-life of a base is 1,000,000 hours, and the hydrolysis half-life of a depurinated base is 400 hours [Friedberg]. Thus, after 1000 hours, approximately 0.1% of bases will be damaged, and 10% of 2500-mer strands will remain unbroken. This last figure is very dependent on the length of the strands; only 0.1% of 5000-mers would survive. While not good, this indicates that for "short" strands, errors due to damage can be compensated for by a mild increase in the volume of DNA used in a computation. Additionally, improved rates may be possible by carefully adjusting solvent salinity, pH, and composition – and again minimizing rough physical handling.

In summary, although there are many serious engineering challenges, we do not see any as being clearly insurmountable.

4 A Stickers Machine Proposal

This section describes the details of one possible machine that implements computation using the stickers model. The machine is a sort of "parallel robotic workstation for molecular computation" in which various robotic and fluid flow apparati are controlled by a central programmable electronic computer. It contains of a rack of many test tubes, a small amount of robotics, some fluid pumps and heaters/coolers and some conventional microelectronics. For each of the operations in the model, we have made a specific choice of physical procedures to implement it. Thus the machine represents one particular realization of many possible variations on the ideas discussed above. The proposal is meant to provoke thought about the engineering issues involved in eventually constructing a molecular computer and not as a serious or viable construction plan.

The workstation stores all DNA which represents information during the computation in so called *data tubes*. Each data tube is a closed cylinder with a nipple connector in either end that allows fluid to flow in or out. Near one end on the inside is a permanent membrane which passes solvent but not stickers or memory strands. This membrane gives a polarity to the data tube: the connector on the end closest to the membrane is the "clean" side while the opposite connector is "dirty". No DNA is ever present on the clean side or in the clean connector. When a data tube is not in use it is held clean side down with all of the DNA in the tube resting on the membrane.

The data tubes (which may be empty) hold either sets of memory complexes or supplies of unbound stickers. Specifically, each set of bit strings has associated with it a data tube which holds the memory strands and annealed stickers representing those strings. Also each bit has associated with it a data tube which contains a supply of stickers corresponding to that bit.

Whenever a new set of complexes is created (e.g. from a separation operation) it is placed in a new data tube. Whenever a set of complexes is destroyed (e.g. from a combination operation) the data tube that used to contain it is discarded (or perhaps vigorously washed and sterilized for reuse).

In addition to data tubes there also exist *operator tubes* of similar external construction but with different internal contents. A "blank" operator tube is merely an empty tube with nipple connectors on each end. A "sticker" operator tube is identical except for a permanent filter on its inside which passes stickers but not memory strands. A "separation" operator tube contains many identical copies of one bit's oligo probe. (There is a different separation operator tube for each bit.) It is designed so that the probes cannot escape from the tube but unbound memory complexes can. For example, the probes might be fastened to solid support (by biotinylating them and using a biotin binding matrix) or to large beads with filters that pass memory strands but not beads. For all of the operator tubes, both ends are considered "dirty". Figure 3 illustrates the data and operator tubes.

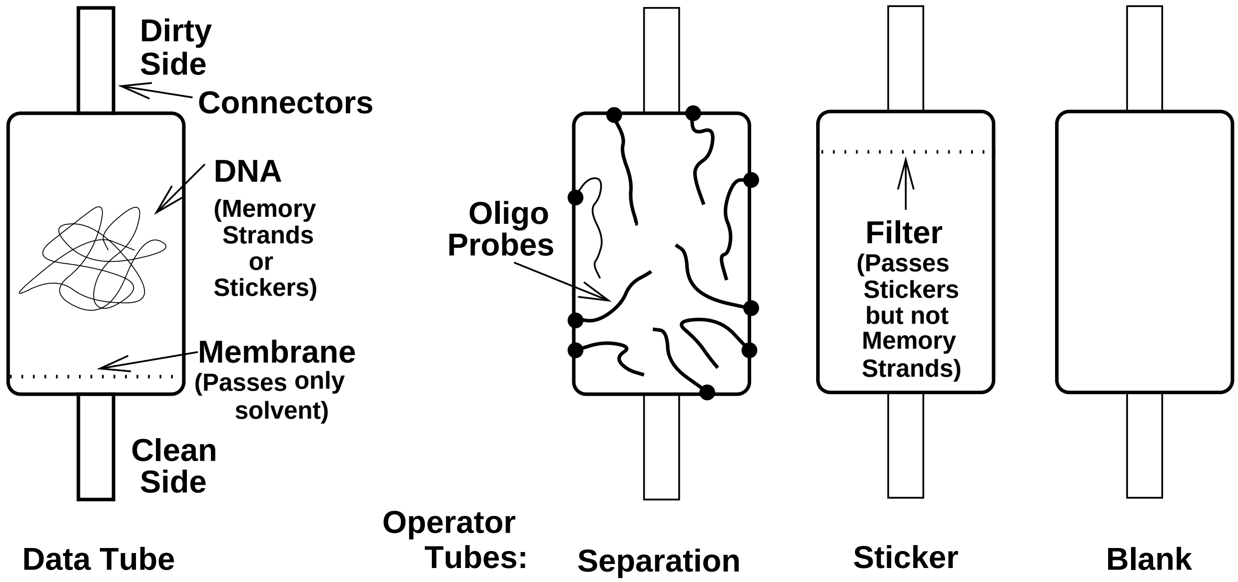

Figure 3: Data and Operator Tubes in the stickers machine.

At any time during the operation of the machine, some tubes are in use and other are not. All tubes that are not in use are stored on a large rack or carousel. Any single operation takes place as follows: under control of the electronic computer two data tubes are selected and removed from the rack by a robot. One operator tube is also selected and removed. The dirty sides of the data tubes are connected to the operator tube, one data tube at each end of the operator. The clean sides of the data tubes are joined by a pump. Solution is cycled through all three tubes. The direction of flow may be towards the first data tube, or vice versa, or both intermingled. The temperature, salinity, direction, and duration of the flow is controlled by the electronic computer. Once the flow stops, one or more of the tubes is disconnected and replaced on the rack (or discarded). New tubes then come in from the rack until there are once again two data tubes and one operator tube and the next operation begins. Notice that in general clean connectors never touch dirty ones and only clean connectors contact the pumping system. This setup for a generic operation is shown in Figure 4.

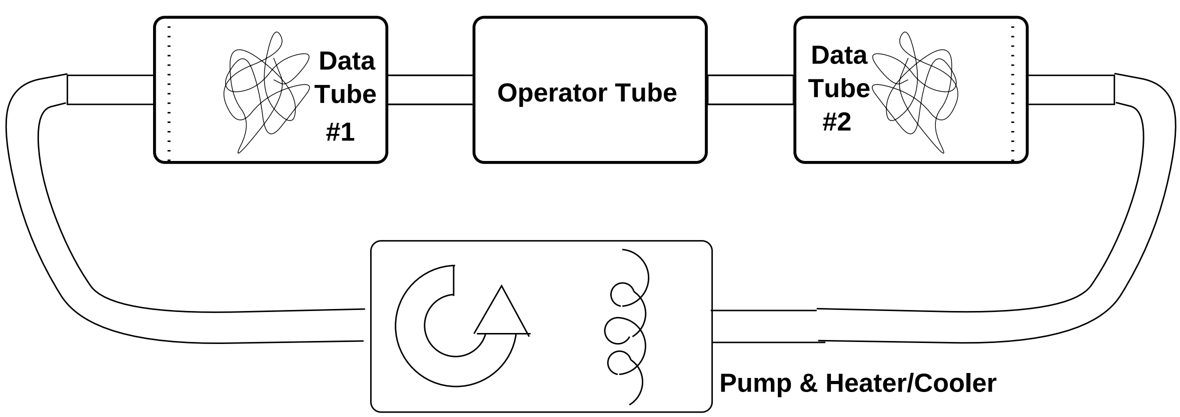

Figure 4: Setup for a generic operation in the stickers machine.

We will now review how each of our conceptual operations can be performed as outlined generically above. The descriptions below are summarized graphically in Figure 5.

- To combine two sets of complexes simply select the two data tubes and a blank operator tube. Cycle cold solution towards (say) the first data tube. This catches all the memory complexes in the first data tube. The second data tube and the blank operator are discarded.

- To separate a set of complexes based on the value of some bit, select the data tube containing the complexes to be separated and also an empty data tube. Select the separation operator tube for the bit in question. Cycle cold solution in both directions for some time; this allows the probes to bind those complexes that have the bit in question off. Next cycle cold solution towards the empty data tube, forcing all the unbound memory complexes into it. Detach this (originally empty) tube and return it to the rack; it holds the complexes with the bit in question on. Replace it with another empty data tube. Cycle hot solution (or perhaps low salinity solution) towards this new data tube. This releases the memory complexes bound to the probes and forces them into the new data tube. Detach this tube and return it to the rack also; it contains complexes with the bit off. Discard the original data tube (now empty) and return the operator tube to the rack.

- To set a bit (add a sticker to a set of complexes), select the data tube containing the complexes and also the data tube containing the sticker supply for the sticker to be added. Using the sticker operator tube cycle cold solution in both directions for some time. This washes the stickers over the memory complexes allowing them to anneal. Now cycle cold solution towards the sticker data tube. This returns the unused stickers and leaves all the memory complexes caught on the filter in the operator tube. Disconnect the sticker data tube and return it to the rack. Replace it with an empty data tube. Cycle cold solution towards the memory complex data tube. This expels the memory complexes from the operator tube and returns them to their data tube. Return the memory complex data tube to the rack and discard the operator tube and empty data tube.

Additional parallelism can be added in many places. For example, setting or clearing bits might be applied to many data tubes at once by stacking all of them after the operator tube. Also, many copies of the robotics might be included to allow several operations to be performed simultaneously (this would also require multiple copies of, for example, the separator operator and sticker operator tubes).

As we have described it, the stickers machine requires relatively rudimentary robotics and electronics. Simple fluid pumps and heaters/coolers are also necessary. It can be stocked with a generic supply of empty data tubes, blank operator tubes, sticker operator tubes, and salt solutions of various concentrations. It contains data tubes containing both the original sets of memory strands and the sticker supplies for each bit. It also needs to be loaded with the separation operator tubes for each bit. An important feature is that these tubes are reusable from problem to problem, so long as the number of bits required does not exceed the number of regions on the designed memory strand. For a problem of reasonable size on the order of a few thousand tubes might be required (for example DES as described in [Adleman3]). With each data tube being a few $m\ell$ in size and operator tubes perhaps a hundred times this size it is not inconceivable that such a machine might fit on a desktop or lab bench. This example directly addresses the concern that any useful or hard computation will require an enormous volume of DNA by demonstrating both a specific problem and a specific machine proposal for which this seems far from true.

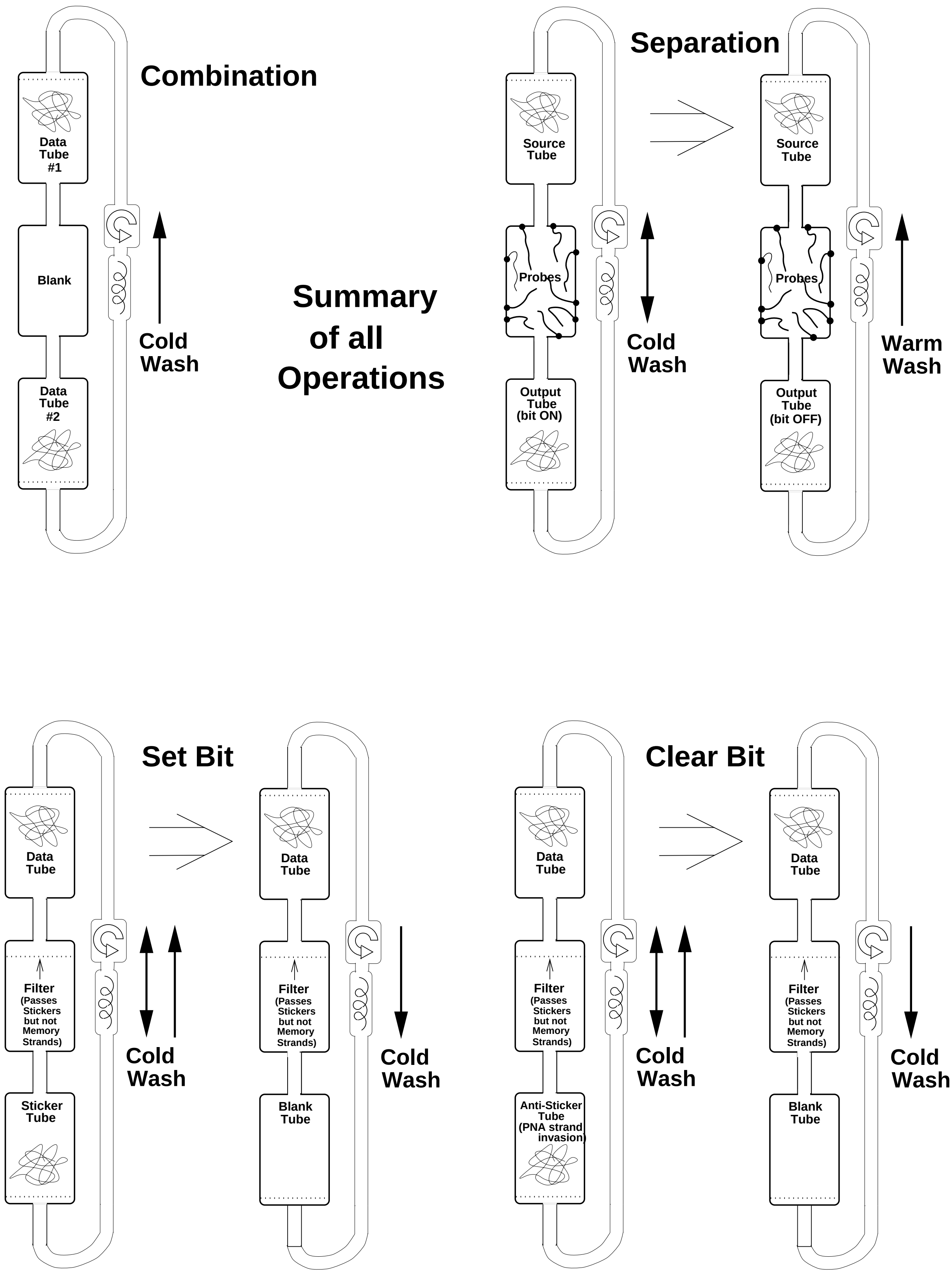

Figure 5: Graphical synopsis of all operations in the stickers machine.

5 Reducing Error Rates: A Refinery Model

In this section we introduce a second possible implementation of the stickers model. In contrast to the "stickers machine" discussed above, the "stickers refinery" addresses the issue of how to perform reliable computation using a very unreliable separation operator. The refinery model also illustrates the principle of pipelining, whereby a large volume of memory complexes can be processed by small capacity operators with minimal slowdown. These advantages come at the cost of a time-space trade-off which we find reasonable.

5.1 An Error Framework

There are three fundamental types of errors that might be made by any molecular computer which attempts to sort a huge library of initial candidate solution complexes into those which encode a solution to a problem and those which do not. It may give some *false positives*, namely some of the complexes that it classifies as solving the problem actually may not. It may also have *false negatives* which occur when complexes that are classified as not solving the problem actually do solve it. Finally, the machine may incur some *strand losses* – some of the complexes which were present in the input may not appear in the output at all: they may simply get lost somewhere inside the machine. What are the error requirements to do useful computation ? It is clear that we want low false positive and false negative rates and few strand losses, but how low do they need to be ?

Our model of a molecular computer is a machine that takes as input a tube encoding a large number of potential solutions to some problem and produces as output two tubes, one labeled *Yes* and the other *No*. In the *Yes* tube are all those complexes which the machine has decided encode solutions to the problem, in the *No* tube are all those complexes which it has decided do not encode solutions. Call a *good* complex one which *actually does* encode a solution and a *bad* complex one which *actually does not*. Because the machine is not perfect, there may be some *good* complexes in the *No* output, some *bad* complexes in the *Yes* output, as well as some losses.

Now we are in a position to state our requirements for error rates: We want two things to be true with high probability (say $1-\epsilon$) each time we run the molecular computer: there is at least one *good* complex in the *Yes* tube *and* the *ratio of good to bad complexes* in the *Yes* tube is reasonable (say ≥ 1). Informally, when we get the answer tube, we will fish around in it, pull out a random complex (if there are any), and read the solution that it encodes. We will be disappointed if either *(a)* we do not find any complexes in the answer tube or *(b)* the complex we read does not actually encode a solution. Our goal is to be disappointed with low probability.

We would like to be able to answer the question: "How good do individual operations have to be for disappointment to be rare ?" Unfortunately, it is very complicated to express the above requirements in terms of conditions on the fidelity of the individual operations such as *separate*. In fact, even for reasonably simple error models, the answers are extremely dependent on the particular architecture of the molecular computer and on the problem being solved[8]. Instead we will work with a model

[8] For example, one could imagine a simple model of errors which is characterized by only three numbers (each between 0 and 1): a false positive rate R_{fp}, a false negative rate R_{fn} and a loss rate R_{loss}. Any given complex is "lost" with probability R_{loss}. If not lost, *good* complexes go to the *Yes* tube with probability $(1-R_{fn})$ and *bad* complexes go to the *No* tube with probability $(1-R_{fp})$ regardless of the specific bit string they encode. Under such a model, if our input tube contains G *good* complexes and B *bad* complexes (typically $G << B$) then we require a false negative rate R_{fn} which is less than some function $f_1(G,B,\epsilon)$, a false positive rate R_{fp} which is less than $f_2(G,B,\epsilon)$, and a loss rate R_{loss} which is less than $f_3(G,B,\epsilon)$, where ϵ is the fraction of runs of the experiment that will result in

which allows us to characterize the *fraction of complexes not yet correctly processed* (denoted simply δ) at some time T after we begin the computation. This quantity can be easily understood as follows: we turn on our molecular computer at time 0 and feed it its input. It works away, placing some complexes in the *Yes* tube and some in the *No* tube. At time T we stop the machine and collect the *Yes* and *No* tubes. At this point, original input complexes fall into three categories: *(1)* those which have been correctly placed[9] into either *Yes* or *No*, *(2)* those which have been incorrectly placed into *Yes* or *No*, and *(3)* those which were either lost or were still being processed by the machine when we turned it off. The fraction of complexes not yet correctly processed (δ) is the fraction of the original input complexes which fall into either categories *(2)* or *(3)* above at time T. We would like $\delta$to be very near zero. Below we develop a model which allows us to compute δ for various machine architectures and also various time and space tradeoff factors in terms of only the fidelity of the atomic operations which are used by the machine, independent of the problem being solved.

5.2 Computing δ

We will consider a very simple mathematical model of a molecular computer as a series of exactly S identical separation operations. The separation operation is used because it is a fundamental operation in the stickers model; both the *set bit* and *clear bit* operations can be described in terms of only separations (see Section 5.7). This model assumes that the algorithm used to process complexes has the effect of passing each one though at most S separations (an assumption which is true for all algorithms that terminate within a known time)[10]. It further assumes that complexes do not interfere with one another, nor do different bit positions on a single strand. For the moment, let us also assume that there are no strand losses; we will return to this crucial issue later.

Assume that (regardless of which bit is being used to separate and of the values of any other bits) each separation operation takes one unit of time to complete and has a probability p of correctly processing each complex in its input[11]. Notice that we expect p to be near unity. In every separation, we assume that each complex ends up in one or the other of the output tubes; no strands are physically lost. Now any computation will take S units of time and when it is done, the fraction of complexes not yet correctly processed will be a depressingly high $\delta = (1 - p^S)$. (For example if $p = 0.9$ and $S = 100$ then $\delta = 0.99997$.) The main point of this section is that *without changing* p (i.e. without improving the basic biotechnology used to implement operations)

disappointment. However, it turns out that even when f_1,f_2, and f_3 have been determined, the conversion from these three numbers to a requirement on the fidelity of individual operations is highly architecture dependent; compare for example the simple *OR* of all bits in a bit string with the simple *AND*.

[9] Note that *good* complexes can be incorrectly processed at some step(s), yet still end up in the "Yes" tube; similarly *bad* complexes can end up in "No" after incorrect processing. We still count these cases as incorrect.

[10] Recall that since "answer readout" and "strand detection" are not permitted during the course of the computation, the algorithm which controls the processing cannot get any feedback and so cannot do any "if then else" type branching. To see that the model assumption is not as restrictive as it may seem, consider architectures that are of the form of feedforward layered circuits with S layers. Each layer receives some number of input tubes from the previous layer and produces some (possibly different) number of output tubes which it passes to the next layer. No tube may go through more than one separation per layer. In this way, for any individual complex such architectures look like a series of S identical separation operations, although different complexes may take different paths through the circuit. The first layer receives as its input the single tube which was the input to the entire problem. The final layer (S) produces as its output the final output tubes for the problem. Any (terminating) algorithm for doing a stickers computation can be converted into a feedforward circuit of this kind.

[11] In practice, operations like *separation* have a much higher probability of correctly processing some inputs than others. For example if hybridization is used, it is much harder for probes to erroneously capture complexes than it is for them to let through complexes which they should capture. All of the mathematics which follows can easily be done for the assymetric probability case although it is somewhat more complicated.

and without reducing S (i.e. without moving to easier problems) the fraction δ can be made much smaller using intelligent space and time tradeoffs.

Imagine that you have in hand enough hardware (i.e. units that perform separations and test tubes) to perform a given computation. A *space tradeoff of factor* H involves obtaining $H-1$ extra identical copies of that hardware, which we may use in parallel. A *time slowdown of factor* M involves taking $M \cdot S$ units of time instead of merely S to perform the computation. How can these factors be used to reduce errors ? Given any algorithm A for performing a computation and factors H and M we would like to investigate algorithm *transformations* which give us a new algorithm A' (that runs in no more than $M \cdot S$ time and requires no more than H copies of the hardware) that has a smaller δ than the original A.

5.3 Repeating the Computation

A basic transformation, *repeating* was proposed by Adleman in [Adleman2]. It makes use of a slowdown factor of M by proposing A' as follows:

- `Repeat` M `times:`
  `Run` A `on input` I`, producing tubes` Y `and` N`.`
  `Discard tube` N `and rename tube` Y `to tube` I`.`
- `Return tube` I `as the ``Yes´´ tube and an empty tube as ``No´´.`

This approach is of value when the original algorithm A was known to very reliably place *good* complexes into its *Yes* output (i.e. low false negatives) but to often also place *bad* complexes into *Yes* (i.e. high false positives). Note that if the original algorithm was known instead to have high false negatives and low false positives then the following version of *repeating* can be used:

- `Make an empty tube` Z`.`
- `Repeat` M `times:`
  `Run` A `on input` I`, producing tubes` Y `and` N`.`
  `Combine tube` Y `into tube` Z`, destroying` Y`.`
  `Rename tube` N `to tube` I`.`
- `Return tube` Z `as the ``Yes´´ tube and tube` $I$ `as ``No´´.`

By how much does *repeating* reduce δ ? The performance of this transformation is bounded by the performance of an imaginary transformation called *repeating with an oracle* which makes use of a new *oracle* operation. The *oracle* takes as input two tubes Y and N and produces as output three tubes: Y',N', and X. In Y' are all the *good* complexes that were in the input tube Y, in N' are all the *bad* complexes that were in the input tube N, and in X are all the *bad* complexes from Y along with all the *good* complexes from N. In other words, the *oracle* "fixes-up" Y and N by putting any incorrectly processed complexes into X. Using this magical operation, *repeating with an oracle* transforms A into the following A':

- `Make an empty tube` Z`.`
- `Repeat` M `times:`
  `Run` A `on input` I`, producing tubes` Y `and` N`.`
  `Run the oracle on` Y `and` N`, producing` Y'`,`N'`, and` X`.`
  `Discard tube` N' `and combine tube` Y' `into tube` Z`, destroying` Y'`.`
  `Rename tube` X `to tube` I`.`
- `Return tube` Z `as the ``Yes´´ tube and tube` $I$ `as ``No´´.`

This transformation improves δ from $1-p^S$ to $(1-p^S)^M$. The vanilla *repeating* transformations can approach but never exceed this improvement. The reason that plain *repeating* works well at all is that for very disparate false positive and negative rates, one can approximate the action of the oracle easily. While these transformations do yield some reduction in δ they require enormous slowdowns to improve even modest sized problems. For larger problems, the slowdowns these transformations require are enormous. Figure 6 shows the slowdown factors required to achieve various performance levels for the case in which $p = 0.9$ and $S = 100$ or $S = 1000$.

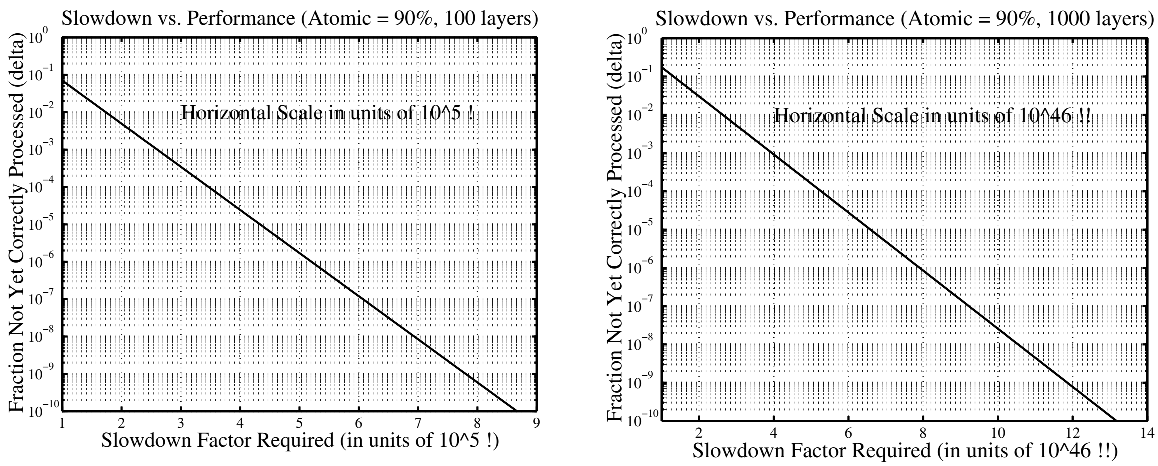

Figure 6: Performance vs. Slowdown for *repetition with an oracle*

5.4 A New Operation: Compound Separation

It is possible to make much better use of space and time tradeoffs than the above transformations do. Shortly, we will develop new transformations which do this, but first we must introduce a new operation which they employ known as *compound separation*.

The central observation is that the following algorithm, analogous to "countercurrent cascade stages" in chemical engineering [Wankat], will exponentially improve upon the accuracy of the *Separation* step:

- Begin with a tube T_0 whose contents we wish to separate based on bit k.
- Begin also with $2N$ extra tubes called $T_{-N}, \ldots, T_{-1}$ and $T_1, \ldots, T_N$, initially empty.
- for t=1 to Q

```
    for j=-N+1 to N-1 s.t.  t+j ≡ 1 (mod 2)
        Separate T_j into T_on and T_off based on bit k
        Combine T_on and T_j+1 into T_j+1
        Combine T_off and T_j-1 into T_j-1
```

(Notice that for odd `t`, odd numbered tubes start off empty and for even `t`, even numbered tubes start off empty.)

Thus each complex will perform a biased random walk in tubes T_{-N} through T_N, with absorption at the boundaries. Most memory complexes which have bit k on will end up in T_N, while most memory complexes which have bit k off will end up in T_{-N}. A graphical illustration of the process is shown in Figure 7. The statistics of such processes have been thoroughly worked out (see the

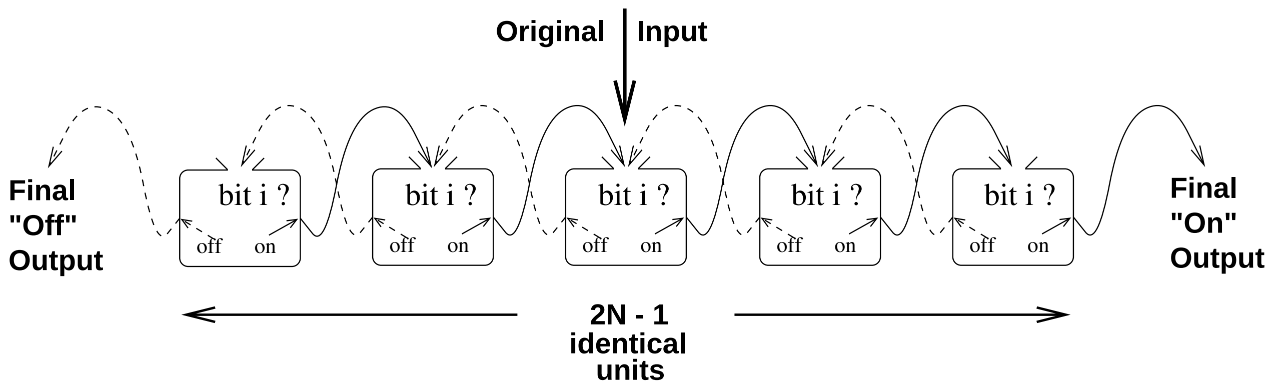

Figure 7: A Compound Separator

"Gambler's Ruin" problem [Feller]). Let p be the probability that a separation step *correctly* moves a complex into T_{on} or T_{off}. At the end of the algorithm, we would like to know the probabilities that a complex with bit k on (or off) will either be in tube T_{-N}, T_N, or still stuck in some other tube. Let us first consider the case $Q = \infty$, i.e. each complex continues to be processed until it it absorbed at either T_{-N} or T_N. Then a complex has probability p_∞ of being correctly processed, where

$$p_\infty = \frac{1}{1 + (\frac{1-p}{p})^N}.$$

For example, if $p = 0.9$, we choose $N = 5$, and then $p_\infty \approx 1 - 10^{-5}$. It is critical to this argument that no memory complexes are lost in the woodwork. However, it is not crucial that Q be ∞. The expected time $t_{compound}$ for a complex to arrive in either T_{-N} or T_N is

$$\langle t_{compound} \rangle = \frac{N}{2p-1}\left(1 - 2\frac{1-r^N}{1-r^{2N}}\right)$$

where $r = \frac{p}{1-p}$. In the example, $\langle t_{compound} \rangle \approx 6.25$. In fact, in this example, $Q = 20$ ensures that fewer than 10^{-4} of the complexes are not correctly processed. Figure 8 shows the performance (δ) of *compound separation* as a function of number of steps (Q) for various chain lengths (N).

We have shown that, by applying the *compound separation* algorithm above, we can achieve excellent error rates even when the fundamental separation operation is not reliable. This comes at the cost of a small linear slowdown (and a few extra tubes) – in general we need to perform $Q \cdot N$ separations instead of one.

Notice that this algorithm can be easily parallelized: if N "atomic" separator units are available instead of just one then the slowdown factor can be reduced to Q by performing all the separations simultaneously (i.e. do all iterations of the inner `for j` ... loop in parallel). We will call this parallelized algorithm *parallel compound separation*.

Although the basic mathematics are not new, to our knowledge the first application of this idea to molecular computation appeared in [Karp]. Their "Super Extract" operation is very similar (although not identical to) the compound separation we have proposed above. We refer the reader to the excellent discussion and detailed analysis (including some interesting bounds) contained therein.

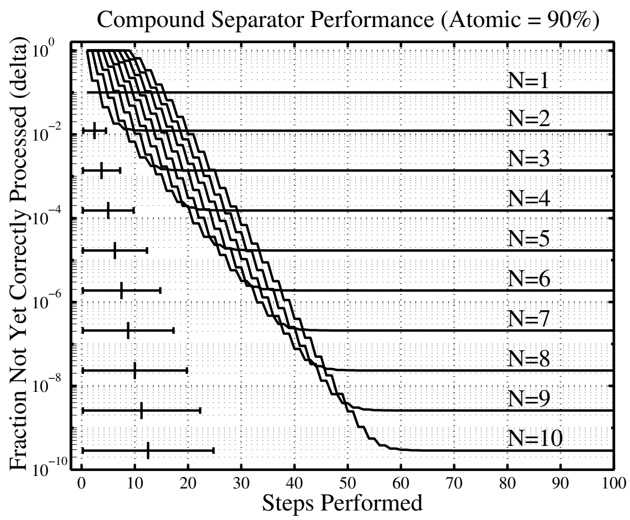

Figure 8: Compound Separation Performance. The bars on the left show the mean time ± one standard deviation for complexes to be absorbed at a boundary.

5.5 Better Transformations: The Refinery Idea

What if we were to replace every *separate* operation in our original algorithm A with a *compound separation* ? This would incur a slowdown factor of $M = Q \cdot N$ but would give an enormous reduction in δ since the fidelity of each separation has improved exponentially. This is exactly the idea behind the *serial refinery* transformation which exploits a slowdown factor of M. It proposes A' to be:

- `Run` A `on input` I`, replacing each` *`separation`* `operation with a` *`compound separation`* `operation of chain size` N `and duration` Q `where` $Q \cdot N \leq M$.

Notice that if a space tradeoff factor of H is also available then we can employ the *one layer refinery* transformation which makes use of the available parallelism H and slowdown M by specifying A' to be:

- `Run` A `on input` I`, replacing each` *`separation`* `operation with a` *`parallel compound separation`* `operation of chain size` N `and duration` Q `where` $Q \cdot N \leq M \cdot H$.

The one layer refinery is so named because if A originally processed one layer in parallel before moving on to the next layer, with sufficient parallelism A' may now process each layer in parallel for Q steps, and then move on to the next layer.

For the moment we defer the issue of how to decide on the optimal factorization of M or $M \cdot H$ into $Q \cdot N$ although we return to it shortly. (The obvious choice is to choose $N = H$ and $Q = M$.) First let us find out how much improvement in δ this transformation buys us. The exact expression for δ is complicated[12] but easily computable. The plots in Figure 9 show the performance (δ) of the *one*

[12] For the afficionado, $\delta = 1 - [\hat{p}(N,Q)]^S$ where $\hat{p}$ is the probability of getting absorbed at the correct boundary in Q steps or less in a biased random walk (bias probability $= p$) with absorption at boundaries N and $-N$. In turn, $\hat{p}(N,Q) = \sum_{i=0}^{Q} \left[\frac{2^i}{2N}(1-p)^{(i-N)/2} p^{(i+N)/2} \sum_{v=1}^{2N-1} \cos^{i-1} \frac{\pi v}{2N} \sin \frac{\pi v}{2N} \sin \frac{\pi v}{2} \right]$ where the expression in square brackets is the probability of absorption in exactly i steps. All of the mathematics can be extended to the case when the random walk bias is different in each direction; see [Feller].

layer refinery transformation as a function of slowdown factor (M) for various compound separator chain lengths (N) and for $S = 100$ and $S = 1000$. The plots assume that we have chosen $Q = M$ and $N = H$.

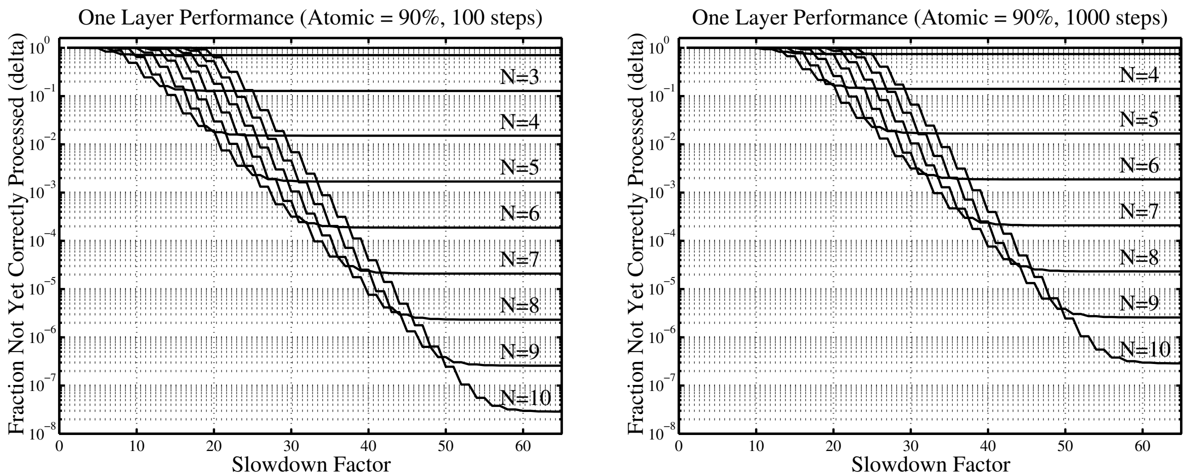

Figure 9: One Layer Refinery performance for S=100 and S=1000. Plots assume that we have chosen $Q = M$ and $N = H$ (see text).

5.6 A Fully Parallel Refinery Architecture

In the remainder of this section we show how, by exploiting the ideas above, a new machine architecture called the "stickers refinery" which achieves the same low error rates as the one layer refinery and greater speed-up by continuously processing all steps in the computation – at the cost, of course, of additional space. The refinery architecture may have other advantages as well, which we will comment on below.

As shown in Figure 8, the mean time for a complex to get through a single compound separation chain is considerably less than the Q required to obtain maximal performance, typically by a factor of about 4. Most of the time during a computation is spent waiting for a few straggling complexes to come out of a separator chain. We can avoid this wasted time by proceeding to process complexes as soon as they are absorbed in T_{-N} or T_N. The *parallel refinery* transformation creates A' by replacing each *separation operation* in A by a *parallel compound separation* of chain length N, and then iteratively processing the entire computation in parallel for T iterations.

Specifically, suppose A has $S \cdot W$ separations (S feedforward layers, at most W per layer) and uses tubes $T^0 \ldots T^J$, where separation i separates $T^{j_{in,i}}$ into $T^{j_{on,i}}$ and $T^{j_{off,i}}$ based on bit k_i. Then the *parallel refinery* transformation given as A' which is defined as:

- Begin with $3 \cdot S \cdot W \cdot (2N-1)$ tubes T_n^j, and $T_{on,n}^j$ and $T_{off,n}^j$ for $-N+1 \leq n \leq N-1$.
- Initially, T_0^1 contains the mother tube complexes.
- for t=1 to T
    - for j=1 to $S \cdot W$ (do all j in parallel)
        - for n=-N+1 to N-1 (do all n in parallel)
            - Separate T_n^j into $T_{on,n}^j$ and $T_{off,n}^j$ based on bit k_j
    - for j=1 to $S \cdot W$ (do all j in parallel)
        - for n=-N+2 to N-2 (do all n in parallel)
            - Combine $T_{on,n-1}^j$ and $T_{off,n+1}^j$ into T_n^j
        - Combine $T_{off,-N+1}^j$ into $T_0^{j_{off,j}}$
        - Combine $T_{on,N-1}^j$ into $T_0^{j_{on,j}}$

Compared to the original A, the fully parallel refinery requires a space tradeoff factor of $H = (2N-1) \cdot S$ (since every separation is expanded) and a slowdown factor (not necessarily integer) of $M = \frac{T}{S}$. The question is, what parallelism H and slowdown $M =$ are required to obtain a desired performance δ? We answer this question by calculating δ given N and T, as before. First we note that the probability that a given complex is correctly processed after T steps can be decomposed into the probability $p_{done}(N, T)$ that it is in either the "Yes" tube or the "No" tube after T steps (i.e. not still in the machine when we stop) and the probability $p_{correct}(N)$ that a complex arriving in a final tube has been correctly processed[13]. Recall that a complex has probability $p_\infty = 1/\left(1 + \left(\frac{1-p}{p}\right)^N\right)$ of having been correctly separated every time it leaves a compound separator, so $p_{correct}(N) = (p_\infty)^S$. The distribution of emergence times can be obtained by convolving the distribution for a single compound separation, thus numerically calculating $p_{done}(N, T)$. Then $\delta = 1 - p_{done} p_{correct}$. The result of doing such a computation for $p = 0.9$, $N = 1 \ldots 10$ and $S = 100$ and $S = 1000$ are shown in Figure 10 below.

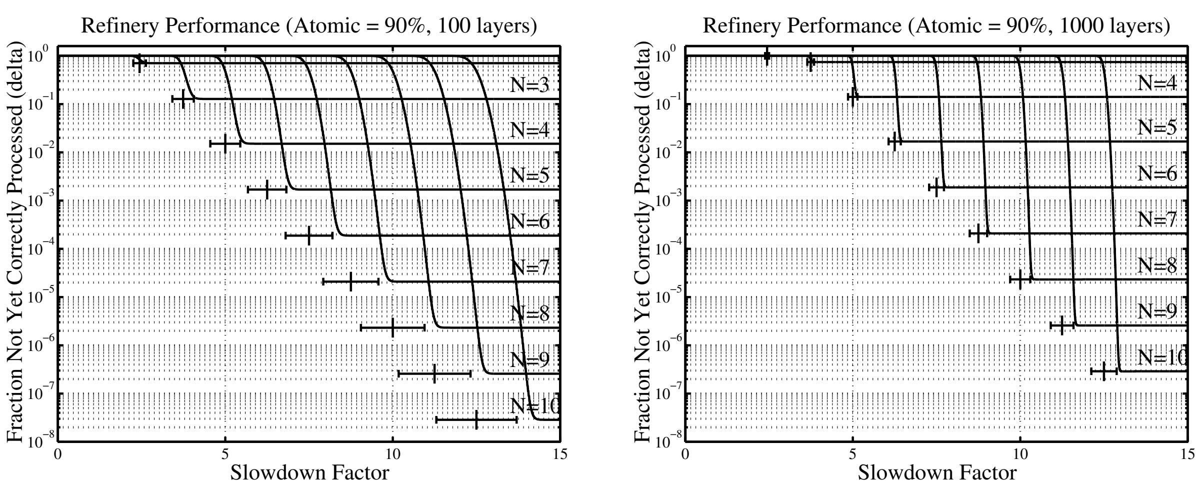

Figure 10: Fully Parallel Refinery performance for S=100 and S=1000. The bars on the left show the mean time $\pm$ one standard deviation for complexes to emerge from the entire refinery.

[13] This second probability is independent of when the complex emerges.

5.7 Advantages of the Full Refinery

With the fully parallel refinery, we can obtain the same target error performance and a roughly 4-fold smaller slowdown factor then the one layer refinery at the cost of S-fold more space and parallelism. This may not seem like a beneficial trade-off since S can be potentially large and 4 is small. In fact, it turns out that the 4-fold speedup can be achieved with an extra space tradeoff of much less than S times[14]. However, the fully parallel refinery affords a number of interesting possibilities. For example, suppose our fundamental separation units can handle limited volume, but we need to process a 10000-fold larger volume of DNA. We can "pipeline" the computation by inputting small aliquots of the mother tube at each step, and waiting until the last aliquot gets out. Now most of the machine is being utilized most of the time instead of idly pumping solution around. If the non-pipelined parallel refinery would have taken 10000 steps, then after about 20000 steps the entire computation will be finished, performing a 10000-fold larger search than the non-pipelined version while taking only twice the time. In other words, we are now exploiting for computation all of the additional parallelism and time employed beyond that used by the naive algorithm, while gaining vastly improved error rates *for free*.

The parallel refinery model does not require re-use of any separation unit to serve at multiple points in the algorithm, and thus a general purpose robotic workstation (such as the stickers machine) is unnecessary. We envision a special-purpose refinery system being assembled, from standard units, for each problem to be solved. A separation unit consists of a reservoir into which complexes are received, an affinity column with DNA probes on solid support, pumps and heaters for the wash and elution, and two exit channels (labeled "on" and "off") which lead permanently (through piping or tubing) to the reservoirs of other separation units. We refer to such a machine as embodying the "stickers refinery architecture". It is our hope that a refinery architecture will alleviate the problem of "lost strands", because the physical permanence of all connections allows temporarily stuck strands to eventually become unstuck and still complete the computation.

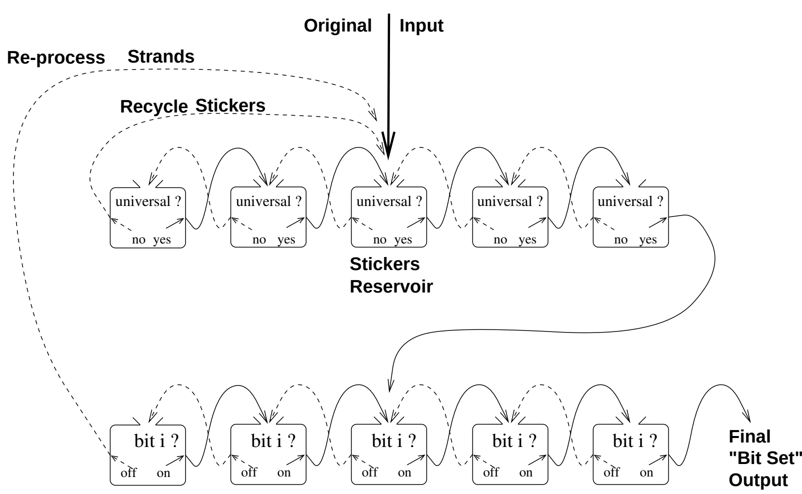

Figure 11: A Reliable *set* Operation

[14]If we consider where the complexes are at some time t, we see that the vast majority of them are near layer $t/\langle t_{compound}\rangle$, leaving the rest of the machine empty – a waste. This observation leads to an intermediate class of refinery algorithms in which a moving window of $L < S$ layers of the circuit are being continually processed as in the full parallel refinery algorithm. Since the distribution of complexes is fairly thin, L can be small, thus requiring less space while achieving nearly identical performance.

Note that the performance of the operations *set* and *clear* can also be improved using these ideas. A *set* operation can be implemented by two compound separations, the first separating based on the universal tag, and the second separating based on the bit being set, as diagrammed in Figure 11. The starting tube is seeded at the beginning of the computation with an excess of stickers, which the universal separation recycles. Complexes which failed to acquire the sticker are returned to the starting tube, where they have another chance to hybridize with a sticker. A similar technique could be used for *clear*, adding a step to purify stickers from PNA clamps.

5.8 Using refineries

It is illustrative to consider using the refinery to solve a particular problem. We will consider breaking DES, for which the naive algorithm A has $S = 6500$ and $W = 32$. Let's suppose $p = 0.9$. Using the one layer refinery algorithm and $N = 10$, we incur a space factor of 19 and a slowdown of ≈ 60 (no further slowdown helps); this achieves $\delta \approx 1.9 \times 10^{-6}$. We started with 2^{56} keys, exactly one of which is good. We can be sure (except for 1.9 in a million) that the good key will end up in the "Yes" tube, but $2^{56}\delta \approx 1.4 \times 10^{11}$ bad keys will be incorrectly processed. Will the incorrectly processed complexes also end up in the "Yes" tube as distractors? In the case of the DES algorithm, we argue that they won't end up in the "Yes" tube [Adleman3]. However, we cannot make the same argument for generic algorithms, and so we consider the worst case scenario in which all of the incorrectly processed complexes are distractors. In this case, we need to achieve $\delta \approx 10^{-17}$ to get the number of distractors below 1. With the one layer refinery, this could either be realized by increasing the space factor to 43 ($N = 22$) and the slowdown to ≈ 125, or by simply re-running the $N = 10$ version mentioned above three times in a row[15] (giving a space factor of 19 and a slowdown of ≈ 180). This last approach is an interesting example of what can be further achieved by *composing* the various algorithm transformations we discussed above.

6 Conclusions

In this paper we have tried to visualize a practical molecular computer. A number of previous concerns [Smith, Hartmanis, Letters to Science] have been addressed. First, it is now clear, from our own work and that of others, that general-purpose algorithms can be implemented by DNA-based computers, potentially solving a wide class of search problems. Second, we now understand that there are challenging problems, such as breaking DES, for which only modest volumes of DNA (e.g. 2 grams) should suffice. Third, we demonstrated that the formation and breaking of covalent bonds is not intrinsic to DNA-based computation. This means that costly and short-lived materials such as enzymes are not necessary, nor are energetically costly processes such as PCR. All the materials in the stickers model are potentially reusable from one computation to the next. Fourth, we have shown that a single essential biotechnology, sequence-specific separation, suffices for constructing a general-purpose molecular computer. Fifth, we now know that separation errors can theoretically be reduced to tolerable levels by invoking a trade-off between time, space, and error rates at the level of algorithm design; we have also illustrated several specific ways in which this can be done and presented encouraging numerical calculations of their performance.

That several major roadblocks have been overcome at a theoretical level suggests that real applications of molecular computation may be feasible in the future. Nonetheless, we emphasize that

[15] Thus the expected number of distractors will be 1.4×10^{11} (first run), 2.7×10^{5} (second run), 0.5 (third run).

substantial engineering challenges remain at almost all stages and that the ultimate success or failure of DNA computing will certainly depend on whether these challenges can be met in laboratory investigations.

7 Acknowledgments

The authors would like to express their appreciation to Professor John Baldeschwieler for his contributions to this paper through early discussions of this work. Sam Roweis and Erik Winfree are also grateful to their advisor, Professor John Hopfield for his perpetual wisdom and long term advice.

References

[Adleman1] Leonard Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science* 266: 1021–1024 (Nov. 11) 1994.

[Adleman2] Leonard Adleman. On Constructing a Molecular Computer. Draft Jan. 11, 1995.

[Adleman3] Leonard Adleman, Paul W. K Rothemund, Sam Roweis, Erik Winfree. On Applying Molecular Computation to the Data Encryption Standard. *These proceedings.*

[Amos] Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant Implementation of DNA Computations. Draft, Jan 1996.

[Baum] Eric B. Baum. DNA Sequences Useful for Computation. Draft 1995.

[Beaver] Don Beaver. Molecular Computing. Penn State University Tech Report CSE-95-001

[Boneh1] Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES Using a Molecular Computer. Technical Report CS-TR-489-95, Princeton University, 1995.

[Boneh2] Dan Boneh, C. Dunworth, R. Lipton, and J. Sgall. On the Computational Power of DNA. Draft 1995.

[Boneh3] Dan Boneh and Richard Lipton. Making DNA Computers Error Resistant. Draft 1995.

[Breslauer] Kenneth J. Breslauer, Ronald Frank, Helmut Blöcker, Luis A. Marky. Predicting DNA duplex stability from the base sequence. *Proc. Natl. Acad. Sci. USA*, 83: 3746–3750, June 1986.

[Castro] Alonso Castro and E. Brooks Shera. Single-molecule Detection: Applications to Ultrasensitive Biochemical Analysis. *Applied Optics*, 34 (18): 3218–3222, June 20, 1995.

[Demidov] Vadim V. Demidov, Michael V. Yavnilovich, Boris P. Belotserkovskii, Maxim D. Frank-Kamenetskii, and Peter E. Nielsen. Kinetics and Mechanism of Polyamide ("peptide") nucleic acid binding to duplex DNA. *Proc Natl. Acad. Sci. USA* 92: 2637–2641, 1995.

[Dempcy] Robert O. Dempcy, Kenneth A. Browne, and Thomas C. Bruice. Synthesis of a thymidyl pentamer of deoxyribonucleic guanidine and binding studies with DNA homopolynucleotides. *Proc. Natl. Acad. Sci. USA*, 92: 6097–6101, June 1995.

[Egholm1] Michael Egholm, Ole Buchardt, Leif Christensen, Carsten Behrens, Susan M. Freier, David A. Driver, Rolf H. Berg, Soeg K. Kim, Begt Norden, and Peter E. Nielsen. PNA hybridizes to complememntary oligonucleotides obeying the Watson-Crick hydrogen bonding rules. *Nature*, 365: 566–568 (7 Oct.), 1993.

[Egholm2] Michael Egholm, Leif Christensen, Kim L. Dueholm, Ole Buchardt, James Coull, and Peter E. Nielsen. Efficient pH-independent sequence-specific DNA binding by pseudoisocytosine-containing bis-PNA. *Nucleic Acids Research*, 23(2): 217–222, 1995.

[Feller] William Feller. *An Introduction to Probability Theory and Its Applications, Volume I, 3rd edition.* John Wiley & Sons. 1968.

[Fredricksen] Harold Fredricksen. A Survey of Full Length Nonlinear Shift Register Cycle Algorithms. *SIAM Review*, 24(2):195-221, 1982.

[Friedberg] Errol C. Friedberg. DNA Repair. W. H. Freeman and Co., 1985.

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

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

[Ito] Takashi Ito, Cassandra L. Smith, Charles R. Cantor. Sequence-specific DNA purification by triplex affinity capture. *Proc. Natl. Acad. Sci. USA*, 89: 495–498, January 1992.

[Karp] Richard Karp, Claire Kenyon and Orli Waarts. Error Resilient DNA Computation. *Laboratoire de l'Informatique du Parallelisme, Ecole Normale Superieure de Lyon*, Research Report Number 95-20 Sept. 1995.

[Kornberg] Arthur Kornberg. *DNA Replication, 2nd edition.* W.H. Freman and Company, New York, New York, 1992.

[Letters to Science] Letters to *Science*, 268: 481–484, 1995.

[Lipton] Richard Lipton. DNA Solution of Hard Computational Problems. *Science* 268: 542–545 (Apr. 28) 1995.

[Meade] *Scientific American* pp. 33–34, May 1995

[Mir] Kalim Mir, personal communication.

[Neveln] Bob Neveln. Comma-free and Synchronizable Codes. *J. theor. Biol.*, 144:209–212, 1990.

[Nielsen] Peter E. Nielsen, Michael Egholm, Rolf H. Berg, and Ole Buchardt. Sequence-Selective Recognition of DNA by Strand Displacement with a Thymine-Substituted Polyamide. *Science*, 254: 1497–1500, 1991.

[Orum] H. Orum, P. E. Nielsen, M. Jorgensen, C. Larsson, C. Stanley, and T. Koch. Sequence-specific Purification of Nucleic Acids by PNA-Controlled Hybrid Selection. *BioTechniques* 19: 472–480, 1995.

[Petruska] John Pestruska and Myron F. Goodman. Enthalpy-Entropy Compensation in DNA Melting Thermodynamics. *Journal of Biological Chemistry* 270(2): 746–750, January 1995.

[Reif] Parallel Molecular Computation: Models and Simulations. Seventh Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA95), Santa Barbara, June, 1995.

[Roos] On the Power of Bio-Computers. Draft Feb. 28, 1995

[Rothemund] Paul W. K. Rothemund. A DNA and Restriction Enzyme Implementation of Turing Machines. To appear.

[Santalucia] J. Santalucia, H. T. Allawi, A. Seneviratne. Improved nearest-neighbor parameters for predicting DNA duplex stability. *Biochemistry* 35(11): 3555-3562, 1996.

[Smith] Warren D. Smith and Allan Schweitzer. DNA Computers in Vitro and Vivo. NECI Technical Report, March 20, 1995.

[Stemmer] Willem P. C. Stemmer. The Evolution of Molecular Computation. *Science*, 270: 1510–1510, 1995.

[Wankat] Phillip C. Wankat. *Separations in Chemical Engineering: Equilibrium Staged Separations.* Elsevier Science Publishing Co., Inc., New York, New York, 1988.

[Wetmur] James G. Wetmur. DNA Probes: Applications of the Principles of Nucleic Acid Hybridization. *Critical Reviews in Biochemistry and Molecular Biology* 36 (3/4) : 227–259, 1991.