JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 6, Number 1, 1999
Mary Ann Liebert, Inc.
Pp. 65–75

On the Reduction of Errors in DNA Computation

SAM ROWEIS and ERIK WINFREE

ABSTRACT

In this paper, we discuss techniques for reducing errors in DNA computation. We investigate several methods for achieving acceptable overall error rates for a computation using basic operations that are error prone. We analyze a single essential biotechnology, sequence-specific separation, and show that separation errors theoretically can be reduced to tolerable levels by invoking a tradeoff between time, space, and error rates at the level of algorithm design. These tradeoffs do not depend upon improvement of the underlying biotechnology which implements the separation step. We outline several specific ways in which error reduction can be done and present numerical calculations of their performance.

Key words: DNA computations, error reduction, errors, molecular computation

1. REDUCING SEPARATION ERROR RATES

IN THIS PAPER, we address the issue of how to perform reliable computation using an unreliable separation operator. The separation operation is a fundamental operation in proposals for DNA based computation. It was originally used as a part of the algorithm for solving the Hamiltonian Path Problem in Adleman (1994); in Lipton (1995), separation was used almost exclusively for solving the Formula Satisfiability problem; in Boneh *et al.* (1996), the algorithm for solving the Circuit Satisfiability problem relies heavily upon the separation operation; and the sticker model of Roweis *et al.* (1998b) shows that separation alone could suffice for making a general purpose molecular computer. Unfortunately, laboratory experience has shown separation to be somewhat unreliable, which has led several researchers to investigate models of computation that do not use the separation operation (Amos *et al.*, 1998; Liv *et al.*, 1998). However, the algorithmic flexibility which the separation operation permits motivates us to investigate ways of making the separation operation more reliable. One approach would be to directly improve the biotechnology. Another approach would be to use redundancy—more DNA—provided by PCR amplification (Boneh and Lipton, 1998). The approach taken in this paper, on the other hand, is to increase the reliability of computation by algorithmic means.

Some of the methods we suggest were proposed independently and analyzed in mathematical detail by Karp *et al.* (1996). Their algorithm for error-resilient bit evaluation is essentially (though not exactly) equivalent to our compound separator. Their work emphasizes the mathematical analysis of the error probabilities, they consider unequal atomic false-positive and false-negative probabilities, and they are able to prove some remarkable upper and lower bounds. Our work, on the other hand, emphasizes and numerically illustrates the application of these ideas at the "practical" level. We also introduce the pipelined, fully parallel refinery model. This model shows that the time and space parallelism required to obtain acceptable error rates may be used simultaneously for performing additional computation.

Computation and Neural Systems Option, California Institute of Technology, Pasadena, California.
Present address: Department of Molecular Biology, Princeton University, Princeton, NJ.

This paper consists of five parts. First, we introduce a framework for discussing errors in separation-based molecular computation. Second, we analyze previous suggestions to reduce errors by repetition, and show these methods to be insufficient. Third, we introduce the *compound separator* and show that reliable computation can be achieved at the expense of a time and space tradeoff. Fourth, with the *refinery model* we introduce 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 tradeoff which we find reasonable. Finally, we discuss how these improvements can be integrated with general-purpose models of computation, such as the sticker model (Roweis *et al.*, 1998b).

2. AN ERROR FRAMEWORK

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 of which is intended to contain valid solutions. Potential solutions (bitstrings) may be encoded into physical molecules (often DNA) in a variety of ways. Whichever encoding is employed, we will use the term *complex* to refer to the molecule(s) which represent an individual bitstring; (for an example, see Roweis *et al.*, 1998b). The *separation* operation takes as input a tube containing many complexes and produces as output two tubes. It attempts to place in one tube all input complexes with a particular bit *on*, and in the other tube those with that bit *off*.

There are three fundamental types of errors that might be made by any molecular computer which attempts, by use of the separation, 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?

Let us label the two output tubes produced by the molecular computer as the *Yes* tube and the *No* tube. 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.[1] Instead we will work with a model which allows us to characterize the expected *fraction of complexes not yet correctly processed* at some time T after we begin the computation. We

[1] 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 $G \ll B$) 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 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 problem dependent; compare for example the simple *OR* of all bits in a bit string with the simple *AND*.

will call this fraction δ. 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[2] 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 expected fraction of complexes not yet correctly processed (δ) is the expected fraction of the original input complexes which fall into either categories *(2)* or *(3)* at time T. We would like δ to be very near zero. Below we develop a model which allows us to compute δ for various error-correction strategies 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.

3. ERROR RATES AND TIME-SPACE TRADEOFFS

For error analysis, we will consider a very simple mathematical model of a molecular computation as a series of exactly S separation operations. This model assumes that the algorithm used to process complexes has the effect of passing each one through at most S separations (an assumption which is true for all algorithms that terminate within a known time).[3] It further assumes that complexes do not interact with one another, nor do different bit positions on a single strand. In developing the results that follow, we do not consider strand losses; techniques for dealing with strand loss by PCR amplification are discussed in Boneh and Lipton (1998).

To see that the model assumption is not as restrictive as it may seem, consider algorithms that are of the form of feed-forward layered circuits with S layers (with *separation* operations at each layer and *combine* operations between 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 algorithms 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 separation-based computation can be converted into a feed-forward circuit of this kind. Thus the error probabilities for the algorithm can be modeled simply by the series of separations.

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 each complex has a probability p of being correctly processed (independent of the other complexes, because we have assumed they do not interact).[4] 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 expected 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 $\delta = 0.99997$. Our main point is that *without changing* p (i.e., without improving the basic biotechnology used to implement operations) 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 we have in hand enough hardware (i.e., units that perform separations and test tubes) to perform a given computation. A *space increase of factor* H involves obtaining $H - 1$ extra identical copies of that hardware, which may be used 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) having a smaller δ than the original A.

---

[2]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.

[3]We will not consider "answer readout" and "strand detection" during the course of the computation, so the machine that controls the processing cannot get any feedback and cannot do any "if then else" type branching.

[4]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 that follows can easily be done for the asymmetric probability case although it is somewhat more complicated (for example, see Adleman, 1996; Karp *et al.*, 1996).

4. REPEATING THE COMPUTATION

A basic transformation, called *repeating,* was proposed by Adleman (1996). 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 either *repeating* 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 1 shows the slowdown factors required to achieve various performance levels for the case in which $p = 0.9$ and $S = 100$ or $S = 1{,}000$.

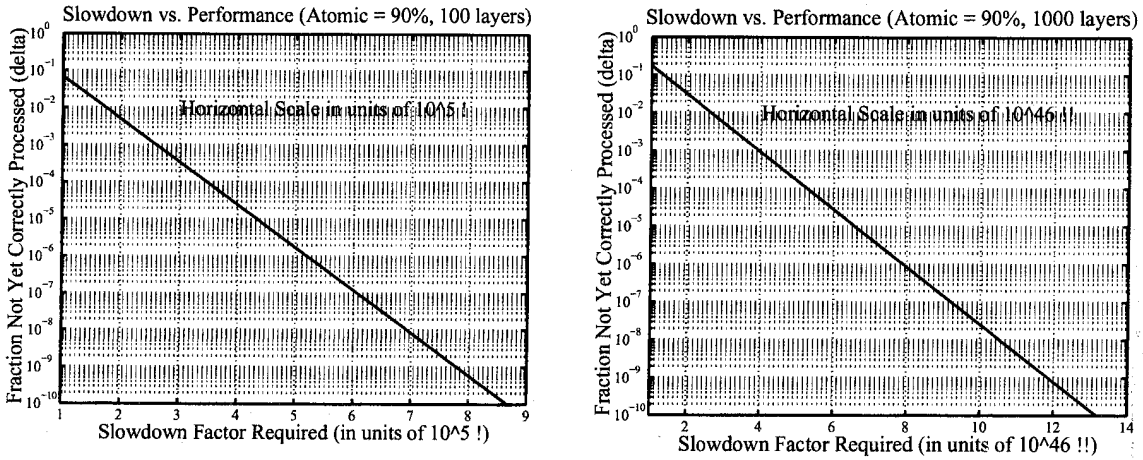

FIG. 1. Performance versus slowdown for *repetition with an oracle.*

5. 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 we call the *compound separation*.

The central observation is that the following algorithm, analogous to "countercurrent cascade stages" in chemical engineering (Wankat, 1988), 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. Therefore at each timestep we need to perform either N or $N-1$ separations.)

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 2. The statistics of such processes have been thoroughly worked out (see the "Gambler's Ruin" problem in Feller, 1968). 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 absorbed at either T_{-N} or T_N. Then a complex has probability p_∞ of being correctly processed (Feller, 1968), where

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

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 permanently lost. However, it is not crucial that Q be ∞. The expected time $\langle t_{compound}\rangle$ for a complex to arrive in either T_{-N} or T_N is (Feller, 1968)

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

where $r=\frac{p}{1-p}$. In the example, $\langle t_{compound}\rangle \approx 6.25$. Not only is the average time small, but the variance is also reasonable. In fact, in this example, $Q=20$ ensures that fewer than 10^{-4} of the complexes are not correctly processed. Figure 3 shows the performance (δ) of *compound separation* as a function of number of steps (Q) for various chain lengths (N).

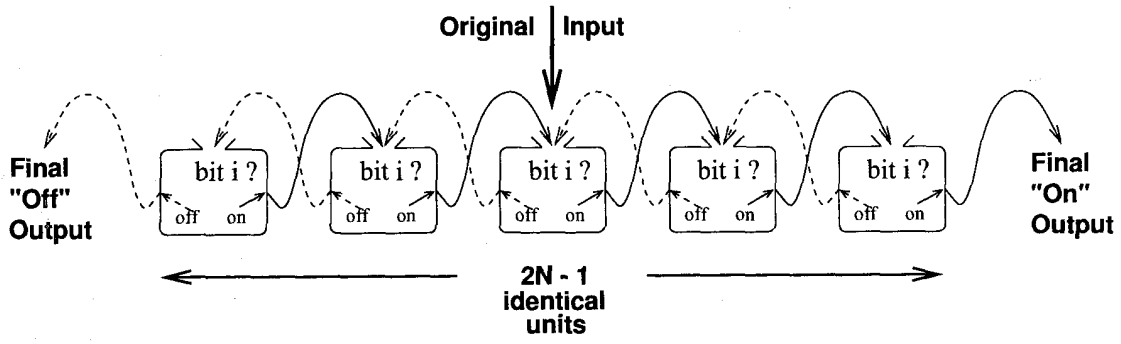

FIG. 2. A compound separator.

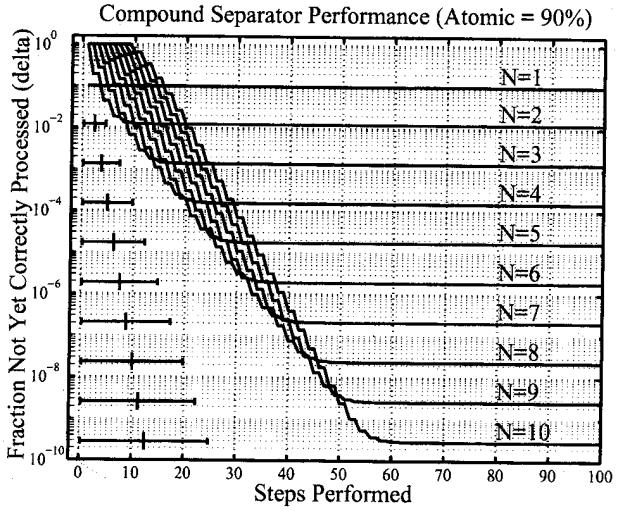

FIG. 3. Compound separation performance. The bars on the left show the mean time ± one standard deviation for complexes to be absorbed at a boundary.

We have shown that, by applying the *compound separation* algorithm above, a linear slowdown (and a few extra tubes) results in an exponential decrease in the error rates, even when the fundamental separation operation is not reliable.

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.*

6. EVEN 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 what we call 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 increase factor of H is also available then we can employ what we call 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 $Q \cdot N \leq M \cdot H$ and $Q \leq M$.

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.

If we are granted parallelism H and slowdown M, there arises the issue of which allowable values of $Q \leq M$ and $N \leq (M \cdot H)/Q$ result in the smallest δ. In this paper, we only consider the choice $N = H$ and $Q = M$. Let us find out how much improvement in δ this transformation buys us. The exact expression for δ is complicated[5] but easily computable. The plots in Figure 4 show the performance (δ) of the *one-layer refinery*

[5]For the afficionado, $\delta = 1 - [\hat{p}(p, N, Q)]^S$ 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, $\hat{p}(p, N, Q) = \sum_{i=0}^{Q} [\frac{2^i}{2N}(1-p)^{(i-N)/2} p^{(i+N)/2} \sum_{v=1}^{2N-1} \cos^{i-1}\frac{\pi v}{2N} \sin\frac{\pi v}{2N} \sin\frac{\pi v}{2}]$ 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 (Feller, 1968).

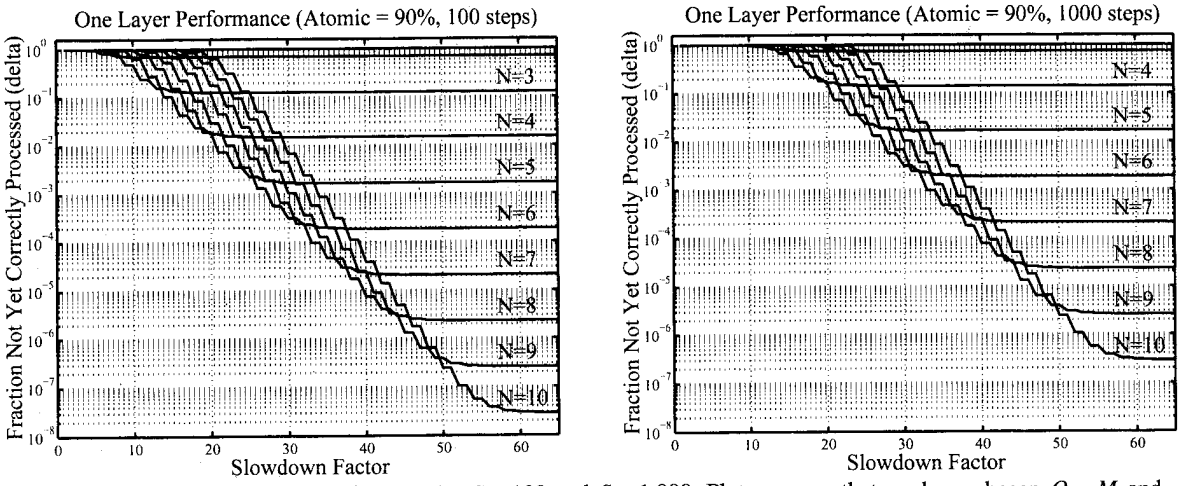

FIG. 4. One-layer refinery performance for $S = 100$ and $S = 1{,}000$. Plots assume that we have chosen $Q = M$ and $N = H$.

transformation as a function of slowdown factor (M) for various compound separator chain lengths (N) and for $S = 100$ and $S = 1{,}000$. The plots assume that we have chosen $Q = M$ and $N = H$.

7. A FULLY PARALLEL REFINERY ARCHITECTURE

In the remainder of this paper, we show how, by exploiting the ideas above, a new transformation can achieve the same low error rates as the one-layer refinery with greater speed-up. The transformation, called the *parallel refinery*, continuously processes *all* steps in the computation—at the cost, of course, of additional space. The refinery architecture has other advantages as well, on which we will comment below.

As shown in Figure 3, 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, by a factor of about 5 in the examples we show. 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 feed-forward layers, at most W per layer) and employs W atomic separators in parallel. The original A 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 defines A' to be:

- 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 $-N+1 \le n \le N-1$.
- Initially, T_0^1 contains the input 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 increase 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

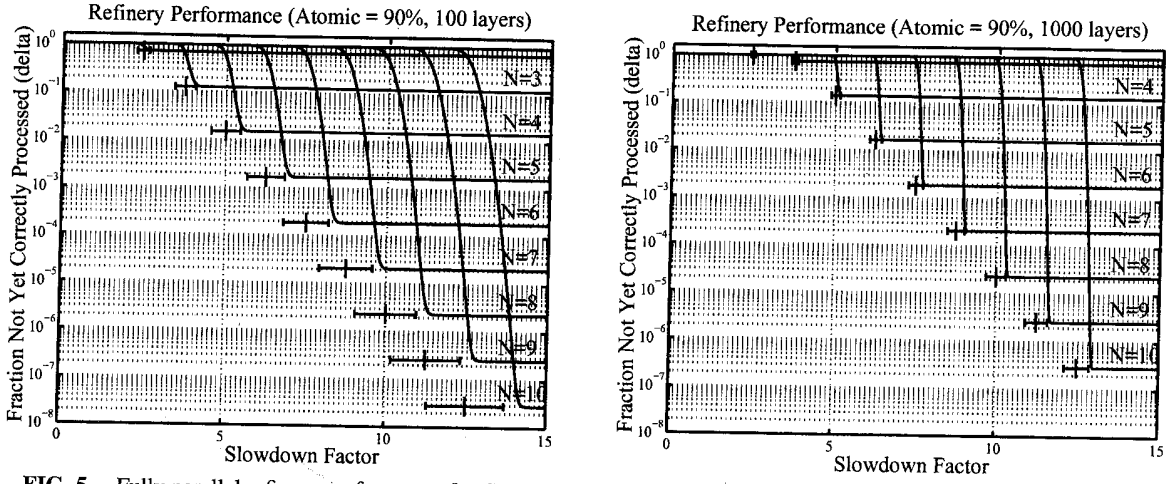

FIG. 5. Fully parallel refinery performance for $S = 100$ and $S = 1{,}000$. The bars on the left show the mean time $\pm$ one standard deviation for complexes to emerge from the entire refinery.

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.[6] Recall that a complex has probability $p_\infty = 1/(1 + (\frac{1-p}{p})^N)$ of having been correctly separated every time it leaves a compound separator, so we conclude that $p_{correct}(N) = (p_\infty)^S$. The distribution of emergence times from the S^{th} layer can be obtained by convolving the distribution for a single compound separation[7] with itself S times, thus numerically calculating $p_{done}(N, T)$. Intuitively (by the Law of Large Numbers), we expect this distribution to have a narrow peak centered at $S \cdot \langle t_{compound} \rangle$ and consequently M is not much more than $\langle t_{compound} \rangle$. Using the formulas for p_{done} and $p_{correct}$, we can calculate $\delta = 1 - p_{done} p_{correct}$. The result of doing such a computation for $p = 0.9$, $N = 1 \cdots 10$ and $S = 100$ and $S = 1000$ are shown in Figure 5.

8. ADVANTAGES OF THE FULL REFINERY

With the full parallel refinery, we can obain the same target error performance and a smaller slowdown factor (roughly fivefold smaller when $p = 0.9$) than the one layer refinery, at the cost of S-fold more space and parallelism. This may not seem like a beneficial tradeoff since S can be potentially large and the speedup in small.[8] The large amount of parallelism is being wasted since most of the hardware is unused most of the time. 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. There are two possible responses to this observation.

One response is to attempt to gain the same speedup using less parallelism. Because most of the strands are near layer $t/\langle t_{compound} \rangle$ we only need enough hardware to process the layers which hold the width of this distribution. This observation leads to an intermediate class of refinery algorithms in which a moving window of $L \ll S$ layers of the circuit are being continually processed as in the full parallel refinery algorithm. Since the spatial distribution of complexes at any time is fairly thin, L can be small, thus requiring much less space while achieving nearly identical performance.

However the other respone is to make better use of the large parallelism required. The key insight is that if we wish to process a large volume of complexes, we can keep the entire machine busy by "pipelining" the computation. For example, suppose our atomic separation units can only handle a limited amount of material, but we need to process a K-fold larger volume. Instead of running the original (or one-layer refinery) algorithm K times, we can input a small aliquot of material into the beginning of the parallel refinery at each timestep. Now most of the machine is being utilized most of the time instead of idly pumping solution around.

---

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

[7] The distribution of emergence times from a single compound separator can be obtained from the formula given in footnote 5: $p_{single}(T) = \hat{p}(p, N, T) + \hat{p}(1 - p, N, T)$.

[8] The numerical value of the speedup (e.g., 5) does depend upon the atomic separation probability p (e.g., 0.9).

We can estimate the amount of material processed using this pipelining strategy. Assume the atomic separators can process no more than C complexes without overflowing. Each complex moves away from the input tube at an average speed of $N/\langle t_{compound}\rangle$ tubes per timestep. Therefore, we can inject an aliquot of $C \cdot N/\langle t_{compound}\rangle$ complexes into the input tube at each timestep. Suppose that a single aliquot can be processed in T timesteps, achieving our target δ. Recall that $T \approx \langle t_{compound}\rangle S$. After $(K+1)T$ timesteps, $K \cdot T$ aliquots will have been fully processed. Thus, we have processed $K \cdot T \cdot C \cdot N/\langle t_{compound}\rangle \approx C \cdot N \cdot K \cdot S$ complexes using $(K+1)T/S$ more time and $(2N-1) \cdot S$ more space than the original error-prone algorithm A would have taken to process a single aliquot of size C. So we have incurred a space-time penalty of roughly $2N \cdot T \cdot K$ but processed $N \cdot K \cdot S$ more material. In other words, the space-time cost of correcting errors is only about $2\langle t_{compound}\rangle$. For comparison, processing C complexes with the one-layer refinery requires a space-time product of roughly $5N \cdot T/S \approx 5N\langle t_{compound}\rangle$. In other words, the pipelined parallel refinery gives a reduction of roughly $\frac{5}{2}N$ in cost. In conclusion, we are now exploiting for computation the additional parallelism and time employed beyond that used by the naive algorithm, while gaining vastly improved error rates.

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 one proposed in Roweis *et al.* (1998b)—is unnecessary. We envision a special-purpose refinery system being assembled, from standard units, for each problem to be solved. For example, a separation unit could consist 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. 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.

Note that the performance of the operations other than separation can also be improved using these ideas. For example, the *set* operation, as described in Roweis *et al.* (1998b), 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 6. 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.

9. CONCLUSIONS

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 $S = 6{,}500$ and $W = 32$. Let's suppose $p = 0.9$. Using the one-layer

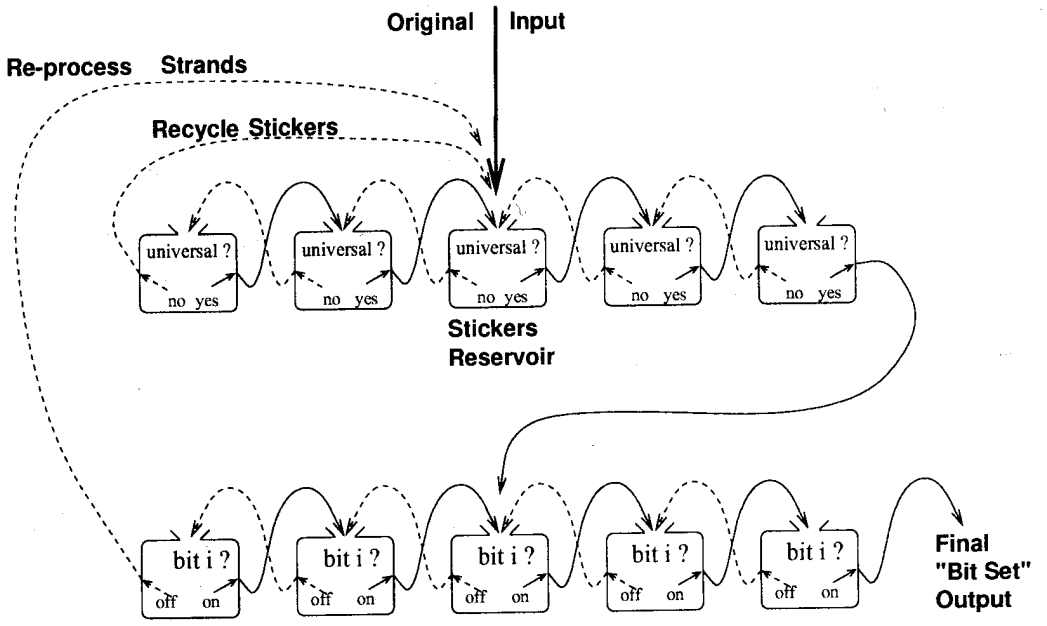

FIG. 6. A reliable *set* operation.

refinery algorithm and $N = 10$, we incur a space increase factor of 10 and a slowdown factor of $\approx$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 (Adleman *et al.*, 1999).

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 $N = 22$ and the slowdown to $\approx$125, or by simply re-running the $N = 10$ version mentioned above three times in a row[9] (giving a space increase factor of 19 and a slowdown of $\approx$180). This last approach is an interesting example of what can be further achieved by *composing* the various algorithm transformations we discussed above, since it uses *repeating*.

We have shown that separation errors theoretically can be reduced to tolerable levels by invoking a tradeoff 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.

ACKNOWLEDGMENTS

We would like to express our appreciation to Professor John Baldeschwieler for his contributions to this paper through early discussions of this work. We are also grateful to our advisor, Professor John Hopfield, for his perpetual wisdom and long-term advice. A preliminary version of this paper previously appeared as Section 5 of Roweis *et al.* (1998a). The MATLAB code used to generate all the figures in this paper is also available by request from *roweis@cns.caltech.edu.* S.R. 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. E.W. is supported in part by National Institute for Mental Health (NIMH) training grant no. 5 T32 MH 19138-06 and also by General Motors' Technology Research Partnership program.

REFERENCES

Adleman, L.M. 1994. Molecular computation of solutions to combinatorial problems. *Science* 266, 1021–1024.

Adleman, L.M. 1996. On constructing a molecular computer. *In* Lipton, R.J., and Baum, E.B., eds., *DNA-Based Computers. DIMACS workshop, April 4, 1995, Volume 27*, American Mathematical Society, Providence, RI, pp. 1–21.

Adleman, L.M., Rothemund, P.W.K., Roweis, S., and Winfree, E. 1999. On applying molecular computation to the data encryption standard. *J. Comput. Biology* 6(1).

Amos, M., Gibbons, A., and Hodgson, D. 1998. Error-resistant implementation of DNA computations. *In:* Landweber, L.F., and Baum, E.B., eds., 1998. *DNA-Based Computers II: DIMACS workshop, June 10–12, 1996, Volume 44*, American Mathematical Society, Providence, RI.

Boneh, D., Dunworth, C., Lipton, R.J., and Sgall, J. 1996. On the computational power of DNA. *Discrete Appl. Math.* 71, 79–94.

Boneh, D., Dunworth, C., Lipton, R.J., and Sgall, J. 1998. Making DNA computers error resistant. *In:* Landweber, L.F., and Baum, E.B., eds., 1998. *DNA-Based Computers II: DIMACS workshop, June 10–12, 1996, Volume 44*, American Mathematical Society, Providence, RI.

Cai, W., Condon, A.E., Corn, R.M., *et al.* 1997. The power of surface-based DNA computation. *In: Proceedings of the First International Conference on Computational Molecular Biology (RECOMB '97).*

Feller, W. 1968. *An Introduction to Probability Theory and Its Applications. Volume 1.* 3rd ed. John Wiley & Sons, New York.

Karp, R.M., Kenyon, C., and Waarts, O. 1996. Error-resilient DNA computation. *In: Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 458–467, AMS/SIAM, Providence, RI.

Landweber, L.F., and Baum, E.B., eds., 1998. *DNA-Based Computers II: DIMACS workshop, June 10–12, 1996, Volume 44*, American Mathematical Society, Providence, RI.

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

Lipton, R.J. 1995. DNA solutions of hard computational problems. *Science* 268, 542–544.

Lipton, R.J. and Baum, E.B., eds. 1996. *DNA-Based Computers: DIMACS workshop, April 4, 1995, Volume 27*, American Mathematical Society, Providence, RI.

Liu, Q., Guo, Z., Fei, Z., Condon, A.E., Corn, R.M., Legally, M.G., and Smith, L.M. 1998. A surface-based approach to DNA Computation. *In:* Landweber, L.F., and Baum, E.B., eds., 1998. *DNA-Based Computers II: DIMACS workshop, June 10–12, 1996, Volume 44*, American Mathematical Society, Providence, RI, pp. 123–132.

Roweis, S., Winfree, E., Burgoyne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., and Adleman, L.M. 1998a. A sticker-based architecture for DNA computation. *In:* Landweber, L.F., and Baum, E.B., eds., 1998. *DNA-Based Computers II: DIMACS workshop, June 10–12, 1996, Volume 44*, American Mathematical Society, Providence, RI, pp. 1–2.

Roweis, S., Winfree, E., Burgayne, R., Chelyapov, N.V., Goodman, M.F., Rothemund, P.W.K., and Adleman, L.M. 1998b. A sticker-based architecture for DNA computation. *J. Comput. Bio.* 5(4), 615–629.

Wankat, P.C. 1988. *Separations in Chemical Engineering: Equilibrium Staged Separations*. Elsevier Science Publishing Co., New York.

Address reprint requests to:
Erik Winfree
Computation and Neural Systems Option
California Institute of Technology
Pasadena, CA 91125

winfree@hope.caltech.edu

Received for publication November 9, 1997; accepted as revised December 13, 1998.