DIMACS Series in Discrete Mathematics
and Theoretical Computer Science

Error Correction in DNA Computing: Misclassification and Strand Loss

Kevin Chen and Erik Winfree

ABSTRACT. We present a method of transforming an extract-based DNA computation that is error-prone into one that is relatively error-free. These improvements in error rates are achieved without the supposition of any improvements in the reliability of the underlying laboratory techniques. We assume that only two types of errors are possible: a DNA strand may be incorrectly processed or it may be lost entirely. We show how to deal with each of these errors individually and then analyze the tradeoff when both must be optimized simultaneously.

1. Introduction

Despite the significant theoretical achievements made in the field of DNA computing, no physical DNA computer has yet been used to successfully solve any real problem. For researchers trying to make DNA computation a physical reality, a central problem is that of error correction since the obvious implementations of DNA algorithms using current lab techniques all produce errors at rates unacceptable for useful computation. In particular, two major sources of errors are misclassification errors caused by faulty extracts and strand loss.

However, the unreliability of the underlying biotechnology does not mean that DNA computing must remain a purely theoretical exercise. Even without improvements in lab techniques, research has shown that error rates can be significantly reduced solely by algorithmic methods [**14, 15**]. In other words, an algorithm for a DNA computer can be mapped to an equivalent one that is comparatively error-free, admittedly at the cost of more time and space. This paper presents such a transformation for extract-based DNA algorithms and an analysis of the tradeoff between the reduction of errors it produces and the extra space and time it requires.

2. Model of Computation

Although many models of DNA computation have been proposed, the canonical model remains the *extract model* of Adleman [**1**], as generalized by Lipton [**2**]. In this model we start with a test tube of DNA strands encoding every possible solution to the problem and we make sure that the sequences are chosen such that the strands will be well-behaved in the laboratory [**16, 17, 18, 19**]. The goal of the computation then is to separate the true solutions (the "good strands") from the non-solutions (the "bad strands").

The algorithm uses only three primitives: *extract*, *combine* and *detect*. An extract step takes as input a test tube T and a subsequence S, and produces as output a *yes-tube* containing the strands in T containing the subsequence S and a *no-tube* containing the other strands. A combine step takes two tubes and produces a single tube containing the contents of both. A detect step is used to check if a test tube contains any DNA strands and, if so, to determine the sequence of one (randomly chosen) strand. We will consider only computations where *detect* is used only at the end of the computation to check for the existence of a solution. Note that since the number of combine steps done is bounded by the number of extracts performed, it is reasonable to use the number of extract steps as our measure of time complexity. If we assume the algorithm divides all strands into a final yes-tube and a final no-tube, then ideally, the yes-tube would contain all the good strands and the no-tube all the bad strands. In a real computation, however, the yes-tube may contain both types of strands, so we must choose a small number of strands from the yes-tube, sequence them and verify their correctness, in effect repeating the detect step several times. By the definition of NP, this verification can be carried out efficiently. There is also a fourth primitive called the *duplicate* step, in which PCR is used to make a single duplicate copy of each strand of DNA in the system. This primitive is not used in the conventional extract-based model but will be introduced later as a way to correct errors in the computation.

Formally, an algorithm in the extract model can be represented as a directed acyclic graph with a source and two sinks. Each node represents a test tube; the source node represents the original tube containing all possible solutions and the two sinks represent the final yes- and no-tubes. All non-sink nodes are labeled with some symbol σ and have two outgoing edges labeled σ and $\overline{\sigma}$. $S(\sigma)$ is the sequence which we use to perform an extract on the node. If more than one edge enter a node, we implicitly do a combine step. We shall assume that at the end of the computation, each strand is in one of the two sink nodes.

Figure 1 shows how we can solve 3-SAT with an extract-based DNA computer; specifically, it illustrates the step that extracts putative solutions that satisfy the clause ($x = 0$ or $y = 1$ or $z = 1$). Starting at the node at the top-left, all strands not containing "x=1" are extracted and put in the yes-tube at the bottom of the figure. Next, from the remaining strands, all those containing "y=1" are extracted and put in the yes-tube, and similarly for the last extract. At the end of this computation, the yes-tube contains all strands satisfying the clause. For 3-SAT computations we can immediately discard all other strands, but in general both the yes- and no-tube strands may be used later on, so we assume that these remaining strands are implicitly kept in a no-tube until the end of the computation. For the next clause, we can do a similar computation but starting with the yes-tube of the previous clause. Clearly, this algorithm requires 3 extract steps and 1 combine step for each clause plus a constant amount of time for the detect step. This gives a running time complexity linear in the size of the formula compared to the exponential time required by the best known algorithm for the classical Turing-machine model. However, an exponential number of DNA strands must be used.

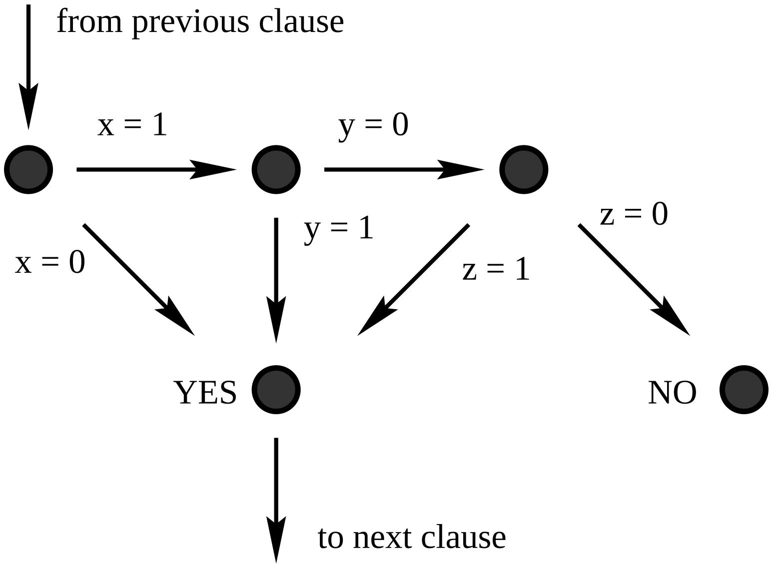

FIGURE 1. Solving 3-SAT with an extract-based DNA computer. Subroutine for "... & ($x = 0$ or $y = 1$ or $z = 1$) & ..."

3. Error model

From the previous section, it is clear that the error rate of the computation is dominated by the error rates of the extract and combine steps. Furthermore, since an algorithm can have at most one combine step per extract, errors that occur in a combine step (loss of DNA strands) will be attributed to the preceding extract. Therefore, from now on, we will assume that errors in processing are caused only by extract steps and we will treat an algorithm as simply a sequence of E extracts. In general the number of extracts performed on each strand may be different. However, if E is the maximum number of extracts performed on any strand, then we can think of E as an upper bound on the number of extracts performed on any strand. We can then transform any algorithm A in which each strand goes through $\leq E$ extracts to an equivalent algorithm A' in which each strand goes through exactly E extracts simply by adding redundant steps. A' is then a layered graph, with E layers. The error rates derived for A' are an upper bound on the error rates for A so from now on we will assume that all strands undergo the same number of extracts.

In this paper, we will consider only two sources of errors in a DNA computation. First, as stated above, each extract step may simply not work correctly, with the result that some strands containing the sequence are classified as not containing it while some strands not containing the sequence are classified as containing it. These errors are called false negatives and false positives respectively. In practice, an implementation of the extract step may have false negative and false positive rates that are several orders of magnitude different. Also, the error rate may be different depending on which DNA sequence is used for the extract since, for example, some sequences bind better to separation beads than others. However, for the sake of simplifying our calculations, we will assume in this paper that both of these error rates are in fact equal. Let the probability of a strand being correctly processed by a single extract step be p, and the probability that the strand is correctly processed by the entire algorithm be $P_{correct}$. For an indication of the overall error rate of a computation where no error correction is used, consider that molecular biologists are generally comfortable with handling around 2^{50} DNA strands at a time. This allows us to reasonably handle an instance of 3-SAT with 50 variables and 200 clauses since this would require 2^{50} strands. To do this we need 3 extracts per clause for a total of 600 extracts, so even with $p = 0.99$, $P_{correct}$ is only $(0.99)^{600} = 0.002$.

The second type of error we consider is caused by a strand being lost during the course of the computation. DNA strands, being physical objects and fragile ones at that, may be lost by getting stuck in the apparatus, by cleavage or cross-linking reactions or by any number of other possible means. Let q be the probability of a strand surviving one step of processing, and let $k = \log_q \frac{1}{2}$. The number of strands will be halved after every k steps; that is, $q^k = .5$. In an extreme case one may have to remove a considerable fraction of the strands after each step for debugging and $k \approx 1$, but we expect that in general $k > 5$ can be achieved. We denote the probability of at least one copy of (perhaps initially many copies of) a strand surviving to the end of the computation as $P_{survival}$.

4. Related Work

Although previous research has been done on improving $P_{correct}$ and $P_{survival}$ individually, to our knowledge no one has taken the next step of studying them together. Intuitively, we need to do more computational steps to improve the rate of correctly processing the strands, but doing so will only increase the chances for a strand to be lost in the computation. If we wish to study the overall rate of error, it is clearly important to study the tradeoff between the two error rates.

In his paper on DHPP [1] Adleman [**1**] briefly mentioned two methods for dealing with errors. For false positives, he suggested that the extract could simply be repeated on the yes-tube, thus weeding out any bad strands that were wrongly misclassified. However, if the no-tube must be used later, repeated separation is useful only if the false negative rate is satisfactory already. So far, this has been shown only for separation techniques that incur substantial strand loss [**20, 21, 22, 23**]. For false negatives – which in his case amounted to strand loss – Adleman suggested using PCR to amplify good (and possibly bad) strands. This is essentially the approach we develop quantitatively in this paper.

Karp et al [**14**] first introduced the notion of a compound extract as a way to deal with processing errors from the extract step. The idea is that one can repeat a faulty extract operation many times to simulate a reliable one. We present essentially the same algorithm but with a more concrete analysis of the construction. Karp claimed that if δ is the desired error rate of the compound extract and $\epsilon = 1-p$ the error rate of the simple extract, then we can achieve the desired error rate by using an extra $O(\log_\epsilon^2 \delta)$ steps, $O(\log_\epsilon \delta)$ of which can be performed in parallel. Our analysis gives a complete proof of the upper bound and shows that the constant factors are in fact not large. Later, Roweis et al [**24, 15**] presented essentially the same algorithm as Karp but with a different analysis.

---

[1] Directed Hamiltonian Path Problem. Adleman's algorithm is sometimes referred to as an algorithm for the *Travelling Salesman Problem* (TSP) although strictly speaking, TSP involves edge-weights and DHPP does not.

Although no algorithm for improving $P_{survival}$ as defined here has been studied, Boneh et al [**25**] proposed and analyzed the use of PCR to reduce errors in decreasing volume computations, a class of DNA algorithms that includes the standard DHPP and 3-SAT algorithms but unfortunately not algorithms for formula-SAT, circuit-SAT or breaking DES [2]. In decreasing volume computations, once a strand is known to not contain a solution, it is discarded, so that the total volume of DNA in the system is decreasing, hopefully at a constant rate. In this case, if bad strands are removed at a faster rate than good strands are lost, then PCR can be used to amplify the good (and bad) strands, making it unlikely for all copies of a good strand to be lost. We adapt Boneh's algorithm to the general problem, where all strands are retained, and we analyze how it affects the probability of a strand surviving the computation. Note that our analysis is not the same as Boneh's because in his case, a bad strand has a much higher probability of being discarded than a good strand, while in our case, both types of strands have equal probabilities of becoming lost.

Although, Karp, Roweis and Boneh are our primary models, it is perhaps worth mentioning a few other error-correcting methods. Amos et al [**26**] proposed an alternative implementation of the extract step which used a restriction enzyme to digest away unwanted strands instead of extracting out the good ones. Ouyang et al [**10**] has successfully implemented this model for a 6-variable SAT problem, and Faulhammer et al [**13**] performed a 9-variable computation using RNA strands and RNAse H as a "universal RNA restriction enzyme." From an algorithmic standpoint the bead-separation and digestion implementations of *extract* are equivalent if *duplicate* is included as a primitive step. An extract step that separates strands into a yes-tube and a no-tube can simulate a digest step by simply discarding the no-tube. Similarly, assuming we are using an encoding (such as Lipton's) where each strand contains either $S(\sigma)$ or $S(\overline{\sigma})$ but not both, we can simulate an extract on sequence S by using a duplicate step to make two copies of the tube, and then digesting the $S(\sigma)$ in one to get the no-tube and digesting $S(\overline{\sigma})$ in the other to get the yes-tube. Although the restriction enzyme digestion approach was proposed to take advantage of the high specificity and efficiency of restriction enzymes, the fact that duplication steps are necessary and that strands may be lost in incidental lab procedures complicates the error analysis. The results in this paper should provide a framework for rigorous analysis of errors in digestion-based DNA computing.

Finally, it is possible for DNA strands to interact with each other in undesired ways and even alter each other's sequences during the duplicate step [**13**]. This paper cannot address these problems; however, the problems may be minimized by experimental protocol modifications and by good choices of coding sequences [**16, 17, 18, 19**]. Good sequences and encodings may also improve the error rates for extracts since some sequences are more likely to be properly extracted than others depending on their biochemical properties [**25**].

[2]The problem is to break the Data Encryption Standard (DES) using a *chosen plain-text attack*, which means that an adversary can obtain a (plain-text, cipher-text) pair where the plain-text is chosen by the adversary.

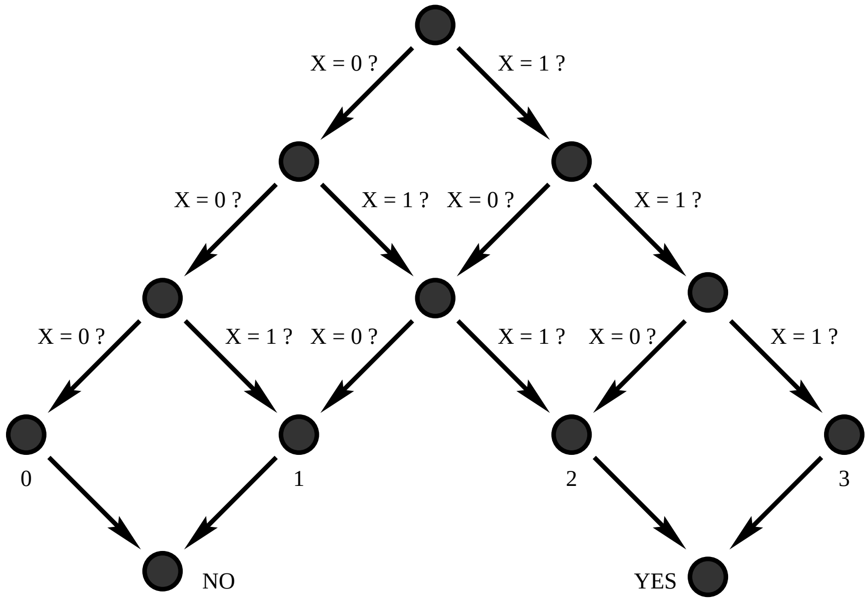

FIGURE 2. The compound extract subroutine, for $n = 3$.

5. Probability of a strand being correctly processed

A compound extract is a *subroutine* designed to be used in place of a simple extract, which we have been referring to up to now simply as an *extract*. A *compound extract* works by repeating the same simple extract many times as outlined in Figure 2, which shows a compound extract on "x=1". Each node represents a test tube and each pair of arrows leaving a node represents a simple extract on "x=1". The question marks have been added to suggest that the simple extract step is not 100% accurate. A compound extract is organized in n phases: in the above example there are three phases each corresponding to a level in the lattice. After each phase, all the strands from the original tube are distributed among the tubes in that level: the j^{th} tube from the left starting at 0 contains strands encoding "x=1" which were extracted correctly exactly j times, as well as strands encoding "x=0" which were extracted incorrectly exactly j times. After the n^{th} phase, we take the right-most tubes and combine them into a single yes-tube, and take the left-most tubes and combine them into a single no-tube. If n is even, and hence after the last phase of extracts there are $n + 1$ tubes, then the central tube may be combined arbitrarily with the yes- or no-tube. The greatest number of tubes needed at any time is n and in principle we can perform all the extracts in a given phase in parallel[3]. Therefore, we measure the time complexity for the compound extract by the number of phases, n.

[3]In fact, we can reduce the number of simple extracts in the compound by a factor of 2 without degrading performance. Observe that the fate of strands leaving the central $\frac{n}{2} \times \frac{n}{2}$ diamond is at that point determined, so all extracts outside the diamond may be eliminated, and all strands may be routed directly to the yes-tube and the no-tube. However, the discussion is simplified by considering the strictly layer phases as described above.

Strands processed according to the Figure 2 perform a random walk. The strands containing the sequence "x=1" have a bias toward the right and those with the sequence "x=0" have a bias (which we assume for this derivation is equal to the rightward bias) towards the left. Therefore the probability of being correctly processed by this method is given by the tail of a binomial distribution. The analysis that follows looks complicated but is in fact straightforward. For a full treatment, see Feller [**27**] Chapter 6.2-3 for the binomial distribution and Chapter 2.9 for Stirling's formula.

We want to find an upper bound on the probability of error due to a compound extract. Define P_e to be this probability and define $P_c = 1 - P_e$. Let

$$b(h; n, p) = \binom{n}{h} p^h (1-p)^{n-h}$$

be the probability that n Bernoulli trials with probabilities p for success and $(1-p)$ for failure result in h successes and $n-h$ failures. The probability of error of the compound extract is given by the sum of the terms of the Binomial distribution from 0 to the central term, $\frac{n-1}{2}$ or $\frac{n}{2}$ (in which case we over estimate either false negatives or false positives). We assume that p is greater than 0.5. Note that the error bound for even $n = 2m$, $P_e \leq \sum_{h=0}^{m} b(h; 2m, p)$, is strictly greater than the error for odd $n = 2m+1$, $P_e = \sum_{h=0}^{m} b(h; 2m+1, p)$, by observing that the strands in the latter tubes are a subset of those that reach the former. Consequently, for mathematical convenience we develop the bound for even n only. To get an upper bound on this quantity, we find an upper bound on the largest, central term, and then bound the rest of the tail with a geometric series.

To get an upper bound on $b(\frac{n}{2}; n, p)$, we use the conventional upper and lower bounds for $n!$ given by Stirling's formula:

$$\sqrt{2\pi n}\; n^n e^{-n} e^{1/(12n+1)} < n! < \sqrt{2\pi n}\; n^n e^{-n} e^{1/(12n)}.$$

Since we assume that n is even,

$$b(\frac{n}{2}; n, p) = \frac{n!}{[(n/2)!]^2} p^{n/2} (1-p)^{n/2}.$$

Plugging the bounds for Stirling's formula in this equation and simplifying (the derivation is tedious but entirely straightforward and hence omitted), we get

$$\begin{aligned} b(\frac{n}{2}; n, p) &\leq \frac{1}{\sqrt{2\pi n}} 2^{n+1} e^{1/(12n) - 2/(6n+1)} p^{n/2} (1-p)^{n/2} \\ &< 2^n p^{n/2} (1-p)^{n/2} \\ &= [4p(1-p)]^{n/2}. \end{aligned} \tag{1}$$

Now to find a good geometric series bound on the tail, consider the ratio between consecutive terms in the tail. If we can bound this ratio with some β, then the probability of an error occuring is

$$\begin{aligned} P_e &= b(\frac{n}{2}; n, p) + b(\frac{n}{2} - 1; n, p) + ... + b(0; n, p) \\ &\leq b(\frac{n}{2}; n, p)[1 + \beta + \beta^2 + ... + \beta^{n/2}] \\ &= b(\frac{n}{2}; n, p)\{1 - \beta^{n/2+1}\}/\{1 - \beta\}. \end{aligned} \tag{2}$$

Typical values and estimates for P_e

	$p = 0.90$ $\approx$	$p = 0.90$ $\leq$	$p = 0.95$ $\approx$	$p = 0.95$ $\leq$	$p = 0.99$ $\approx$	$p = 0.99$ $\leq$
$n = 6$	1.6×10^{-2}	1.4×10^{-1}	2.2×10^{-3}	2.1×10^{-2}	2.0×10^{-5}	1.9×10^{-4}
$n = 10$	1.6×10^{-3}	1.8×10^{-2}	6.4×10^{-5}	7.4×10^{-4}	2.4×10^{-8}	2.9×10^{-7}
$n = 20$	7.2×10^{-6}	1.1×10^{-4}	1.1×10^{-8}	1.8×10^{-7}	1.7×10^{-15}	2.8×10^{-14}

TABLE 1

For $h \leq \frac{n}{2}$ and $p > .5$, the ratio between consecutive terms is

$$\frac{b(h-1;n,p)}{b(h;n,p)} = \frac{h(1-p)}{(n-h+1)p} \leq \frac{(n/2)(1-p)}{(n/2+1)0.5} < 2(1-p). \tag{3}$$

Thus we get our upper bound on the ratio of consecutive terms, $\beta = 2(1-p)$. From (1), (2) and (3) we can get an upper bound on the probability of error as follows:

$$\begin{aligned} P_e &\leq b(\frac{n}{2};n,p)\frac{1-\beta^{n/2+1}}{1-\beta} \\ &< [4p(1-p)]^{n/2}/(2p-1). \end{aligned}$$

And from this we see immediately that for $p > 2/3$,

$$P_e < 3[4p(1-p)]^{n/2} \qquad \text{and} \qquad P_c \geq 1 - 3[4p(1-p)]^{n/2}. \tag{4}$$

So we have shown that using the compound extract method with n phases gives us a compound extract with an error rate decreasing exponentially in n, at the cost of requiring n-fold more time and n-fold greater parallelism. Table 1 shows values for P_e given typical values of n and p, as estimated numerically from the tail distribution and from the analytical bound (4).

Our analysis thus far shows the error rate achievable for a given number of phases, n, of the compound extract. However, for a given target error rate, δ, we would like to find the minimum n that gives us this error rate. Introducing $\epsilon = 1-p$, we can write (4) as

$$P_e \leq 3[4p\epsilon]^{n/2}.$$

Using $n > 2\log_{4p\epsilon} \delta/3$ ensures that $P_e \leq \delta$. When p is close to unity, our choice for n is roughly $2\log_{4\epsilon} \delta/3 = O(\log_\epsilon \delta)$. This is the result Karp presented; our derivation shows, in addition, that the constant factors inherent in the big-O notation are small.

6. Probability of survival

The algorithm is simply to apply one cycle of PCR to all the tubes each time the number of DNA strands falls by half. We assume that exactly one duplicate copy of each strand of DNA is produced. Since we are assuming that strand loss occurs at a constant rate, we need to apply PCR once every k steps for some constant k. If k is not an integer, then the PCR should be applied enough times to keep the volume of DNA constant on the average over the course of the computation. For example, if $k = 4.2$, PCR should be applied after the 4th, 8th, 12th, 16th and 21st steps. For our analysis, however, we will simply take the floor function of k – equivalent to decreasing the stepwise survival probability q. The true error rate will be better than our estimate. Therefore from now on, we will assume that k is an integer.

We will follow a single DNA strand through the computation and use a branching process to analyze P_s, the probability that at least one copy of that strand survives the computation. For more on branching processes, see Feller Chapter 12.3-4. Briefly, a branching process describes particles which are able to reproduce (for example, atoms in a nuclear reaction or, more to the point, DNA strands in a DNA computation). For our algorithm we take a generation to be the number of extract steps required to make the number of strands drop in half, which means that at the end of each generation we do a PCR step. An algorithm with E layers has $\lfloor E/k \rfloor$ full generations.

Let Z_m be the number of strands in the m^{th} generation. Then starting with a single strand as generation zero, let $P_{m,i}$ be the probability that $Z_m = i$. $P_{m,i}$ characterizes the loss of strands and the action of PCR. For example, if PCR perfectly duplicates each strand exactly once, $P_{1,0} = 0.5, P_{1,2} = 0.5$ and $P_{1,i} = 0$ for $i \neq 0, 2$. That is, each strand has probability 0.5 of being lost during any one generation (in which case it produces 0 strands in the next generation, i.e. $P_{1,0} = 0.5$) and it has probability 0.5 of surviving (in which case it is duplicated and produces 2 strands in the next generation, i.e. $P_{1,2} = 0.5$).

We now introduce the notion of a generating function, which is simply a way of expressing a discrete probability distribution in algebraic form. The generating function, $G_m(s)$, of a probabilty distribution is defined to be

$$G_m(s) = \sum_{i \geq 0} P_{m,i} s^i$$

where the dummy variable, s, is simply a place holder without any meaning of its own.

Suppose there were t strands on the first generation. Then the number of strands in the m^{th} generation can be expressed as the sum of t random variables, $X_1, \cdots, X_t$, each representing the number of descendents produced by a particular strand in the first generation. By assumption, X_i all have the same probability distribution and therefore the same generating function. This generating function is just $G_{m-1}(s)$ because X_i are the result of similar branching processes that started at generation one instead of generation zero. Now the probability distribution for the m^{th} generation is given by

$$Prob(Z_m = j) = \sum_{t \geq 0} Prob(Z_1 = t) Prob(X_1 + ... + X_t = j | Z_1 = t). \tag{5}$$

For a fixed t, the distribution of $Z_m = X_1 + \cdots + X_t$ is the convolution of the distributions for $X_1, \ldots, X_t$. Since all X_i have the same distribution $G_{m-1}(s)$, the generating function of $Z_m = X_1 + \cdots + X_t$ is $[G_{m-1}(s)]^t$. We can see this by observing that when we multiply two generating functions, $A(s) = \sum a_i s^i$ and $B(s) = \sum b_i s^i$, term-by-term and collect terms with equal powers of s, the coefficient of s^r is $a_0 b_r + a_1 b_{r-1} + ... + a_r b_0$, which is exactly the r^{th} term in the convolution of $A(s)$ and $B(s)$. Therefore $G_m(s)$ is given by

$$
\begin{aligned}
G_m(s) &= \sum_{j\geq 0} Prob(Z_m = j)s^j \\
&= \sum_{j\geq 0}\sum_{t\geq 0} Prob(Z_1 = t)Prob(X_1 + ... + X_t = j|Z_1 = t)s^j \\
&= \sum_{t\geq 0} P_{1,t} \sum_{j\geq 0} Prob(X_1 + ... + X_t = j|Z_1 = t)s^j \\
&= \sum_{t\geq 0} P_{1,t}[G_{m-1}(s)]^t.
\end{aligned} \tag{6}
$$

But the right hand side of this equation is just the generating function for G_1 substituting $G_{m-1}(s)$ for s. Therefore

$$G_m(s) = G_1(G_{m-1}(s)). \tag{7}$$

Recall that $G_m(0) = P_{m,0} = Prob(Z_m = 0)$ is the probability of the strand being extinct in the m^{th} generation. By (7) we have

$$P_{m,0} = G_1(P_{m-1,0}).$$

Given a particular distribution $G_1(s)$, this polynomial recurrance can be solved to find how quickly (or slowly) $P_{m,0}$ approaches 1.

Here we find bounds for the example discussed above, where PCR is assumed to perfectly duplicate each surviving strand: $P_{1,0} = P_{1,2} = 0.5$, and all other values are zero. Therefore,

$$G_1(s) = \sum_{i\geq 0} P_{1,i}s^i = 0.5 + 0.5s^2$$

and the recurrence is

$$P_{m,0} = G_1(P_{m-1,0}) = 0.5 + 0.5P_{m-1,0}^2.$$

If we take Q_m to be the probability of a strand surviving (i.e. $Q_m = 1 - P_{m,0}$) we get

$$Q_m = 0.5 - 0.5(1 - Q_{m-1})^2 = Q_{m-1} - 0.5Q_{m-1}^2. \tag{8}$$

Now all that remains is to find a lower bound on the recurrence (8). We will show by induction that $Q_m > 1/m$ for $m \geq 4$. For the base case, we compute the first few values of Q_m and $1/m$ by hand: $Q_1 = .5$ and $Q_4 > 1/4$.

For the inductive step, suppose $Q_{m-1} > 1/(m-1)$. Then

$$Q_{m-1} - 0.5Q_{m-1}^2 > 1/(m-1) - 0.5[1/(m-1)]^2$$

if the function $f(Q) = Q - 0.5Q^2$ is strictly increasing. Taking the derivative of $f(x)$, we have $f'(Q) = 1 - Q$ which is positive if $Q < 1$. Since the Q_m's are all probabilities, they are all strictly less than 1, so f is strictly increasing on this interval. This gives us

$$
\begin{aligned}
Q_m &= Q_{m-1} - 0.5Q_{m-1}^2 \\
&> 1/(m-1) - 0.5[1/(m-1)]^2 \\
&> (m^2 - 2m + 1)/[m(m-1)^2] \qquad \text{for } m > 2 \\
&= 1/m
\end{aligned} \tag{9}
$$

which proves that $Q_m > 1/m$ for $m \geq 4$. Similar arguments show that $\frac{1}{m+1} \leq Q_m \leq \frac{2}{m+2}$ for all m; thus our bound is reasonably tight. For convenience, we use $Q_m > \frac{1}{m-1}$ for $m \geq 7$. By our assumption that one generation consists of k extract steps (the final generation may have fewer), we have $m = \lceil nE/k \rceil \leq nE/k + 1$ for the entire computation, so the probability of a strand surviving to the end of the computation with at least one copy remaining is (for $m \geq 7$)

$$P_s > \frac{1}{m-1} > \frac{k}{nE}. \tag{10}$$

This suggests that, if we were to start with *many* copies of each sequence, the probability that *at least one* of the original strands will have descendents in the final tubes will be reasonably high. We show this in the next section.

7. Combining survival and correctness probabilities

To complete the computation, we select a small number of strands from the final yes-tube at random, sequence them and verify them for correctness. Our goal is to guarantee with high probability that there is at least one good strand in the final yes-tube and that the ratio of good strands to bad strands in the yes-tube is high. For concreteness, we show that we can guarantee a ratio of at least 1:1, in which case we should only need to sequence a small number of strands to find a solution. Our terms of merit, therefore, are $P_{correct}$, the probability that a strand that survives the entire computation was processed correctly by each compound extract, $P_{survival}$, the probability that at least one copy of a given sequence survives the entire computation, and $\langle N_{bad} \rangle$, the expected number of bad strands in the final yes-tube.

Unfortunately, the expected number of bad strands in the yes-tube is highly dependent on the particular algorithm. For example, for arbitrary Boolean formulas, it is possible that a single extract error can make a bad strand end up in the final yes-tube. On the other hand, for a practical problem like breaking DES, many extract errors typically would have to be made before a bad strand will arrive in the final yes-tube. To resolve this issue, we will follow Boneh et al [**25**] in requiring that each DNA algorithm A come with information $|M_i|$, where M_i is the set of input sequences that could end up in the yes-tube if it were to suffer i steps of incorrect processing (in the worst-case). By definition, $|M_0|$ is the total number of good sequences and we will assume the worst case that $|M_0| = 1$. Note that a single strand can occur in many M_i.

Our criterion can now be stated as follows: given an algorithm A (and hence E and $|M_i|$) and error rates p and q (and hence k) for correctness and survival, we would like to choose n, the number of phases in the compound extract, and R, the number of redundant copies of each sequence in the original input tube, such that:

1. Each strand is probably correctly processed: $P_{correct} > 0.95$.
2. Each original sequence probably has at least one representative in the final tubes: $P_{survival} > 0.95$.
3. There is probably no more than one bad strand in the yes-tube: $Prob(N_{bad} \leq 1) > 0.95$.

We can now find conditions on n and R that guarantee that each clause be satisfied. First, we note that

$$P_{correct} = (P_c)^E = (1 - P_e)^E > 0.95 \Rightarrow P_e < 1 - 0.95^{1/E}.$$

Thus, the bound (4) on P_e guarantees that #1 is satisfied if

$$n > 2\log_{(4p\epsilon)}[(1-0.95^{1/E})/3].$$

The threshold value of n depends on p and E but not on q, R, or $|M_i|$.

Next, we use $(1-\frac{1}{x})^x < \frac{1}{e}$ for $x > 0$ to obtain from (10) that,

$$P_{survival} = 1-(1-P_s)^R > 1-e^{-P_s R} > 1-e^{-Rk/En}.$$

Thus, #2 is satisfied if

$$R \geq \frac{3En}{k}.$$

The threshold value of R depends on q and E but not on p or $|M_i|$, and it is linear in n. Criteria #1 and #2 delineate a region of possible values with a corner at n_{min}, R_{min}.

Finally, note that if $N_{bad} \geq 2$ at least $1/20^{th}$ of the time, then the expected value $\langle N_{bad}\rangle \geq 1/10$. Thus, $\langle N_{bad}\rangle < 1/10$ would imply that #3 is satisfied. Furthermore, we can get an upper bound on $\langle N_{bad}\rangle$ by presuming that *any* i errors routes the strands in M_i into the yes-tube.

$$\begin{aligned}\langle N_{bad}\rangle &\leq b(1;E,P_e)|M_1|R + b(2;E,P_e)|M_2|R + \cdots + b(E;E,P_e)|M_E|R \\ &< \sum_{i\geq 1}\binom{E}{i}P_e^i|M_i|R \doteq \gamma(n,R). \end{aligned} \tag{11}$$

Thus, #3 is satisfied if $\gamma(n,R) < 1/10$; this final constraint imposes an additional boundary in the space of possible n and R. The shape of this boundary depends the values of $|M_i|$, which can take on almost arbitrary values, but nonetheless certain properties can be predicted. Suppose $\gamma(n_{min}, R_{min}) > 1/10$. Note that simply increasing n decreases all P_e^i exponentially, while the concomitant increase in R is only additive. Therefore, the desired values for n and R are easily found.

Consider the worst case for a b-bit satisfiability problem, where all $2^b - 1$ bad strands are in M_1. Equation (11) reduces to

$$\langle N_{bad}\rangle \leq EP_eM_1R \leq 9E^2[4\epsilon]^{n/2}2^b n/k$$

and therefore, $n = -2\log_{4\epsilon}(9E^2 2^b/k/1000)$ satisfies #3 so long as $n < 100$. In particular, for the 50-variable, 200 clause 3-SAT problem discussed earlier, where $p = 0.99$, no strand loss, and no error correction resulted in $P_{correct} = 0.002$, we can now state that even with $q = 0.90$ (thus $k = 6$), $n = 31$ results in $P_e = 2.6 \times 10^{-24}$ (satisfying #1 with overkill); $R = 9300$ ensures #2 is satisfied; and altogether, $\langle N_{bad}\rangle \leq EP_e|M_1|R = 0.016 < 1/10$, satisfying #3. If each strand is 2000 bases long, then the total population of 2^bR strands weighs a substantial but not outrageous 12 grams.

In contrast, consider the plaintext-ciphertext attack on DES considered by Boneh et al [**3**] and Adleman et al [**28**], for which $b = 56$. Surprisingly, for reasonable assumptions the average number of bad strands in the yes-tube is simply $\langle N_{bad}\rangle = R/256$, independent of the error rate [**28**]. For $E = 6719$, $p = 0.99$, and $q = 0.99$ (thus $k = 69$), we can satisfy #1 using $n = 7$, for which $P_e = 3.4 \times 10^{-7}$ and $P_{correct} > 0.997$; $R = 2048$ satisfies #2 with $P_{survival} > 0.95$; and altogether, $\langle N_{bad}\rangle \approx R/256 = 8$, so sequencing 26 strands from the yes-tube should ensure that the good strand is sequenced and identified. If each strand is 10,000 bases long, then the total population of 2^bR strands weighs 810 grams – substantially better than the 23 Earth masses estimated in [**28**] for $pq = 0.99$.

8. Discussion

The essential point of this investigation is that strand loss and misclassification errors do not present insurmountable obstacles to DNA-based computation. Even in the presence of what would be unthinkable error rates for electronic computers – say 1% per-step error rates – DNA computation can succeed, at the cost of only a modest increase in the number of extract steps and the volume of DNA being handled. 1% error rates are within reach for current biotechnology. Of course, the best solution, if it can be done without affecting the time or cost of each step, would be to improve the fundamental biotechnology of the extract and combine steps directly. Even with such advances, however, the overall error rates required for problems like SAT are unlikely to be achieved without error-correcting algorithms such as the ones discussed here.

Several of the assumptions we used in our derivations, chosen to simplify the mathematics, can be relaxed. One is our assumption that PCR is error-free. It is clear that PCR can make errors by simply copying a strand incorrectly or by failing to make a duplicate copy of a particular strand at all. To account for this, our branching process analysis can be extended to any distribution of values for the P_i's. We can also adapt our algorithm to account for strand loss from PCR by integrating this loss rate into the probability of loss due to extract steps.

The assumption that the combine step is error free can be weakened to the assumption that each combine step has a constant probability of strand loss. Then, just as we can add the strand loss rates of PCR and the extract step, we can also add to their sum the error rate from the combine steps. The assumption that the rate of false-negatives and false-positives are equal is also not necessary and the exact same analysis will hold for this case; Karp et al [**14**] and Adleman [**29**] have shown a simpler variety of compound extract whose false-postive and false-negative rates are both roughly equal to the minimum of the original rates.

However, some non-idealities are not so easy to rectify using our approach. A good example would be *systematic bias* due to sequence-dependent biophysical or biochemical factors – for example, certain DNA sequences may extract with unusually low or high rates, may be amplified particularly reliably or unreliably, or may be lost due to hydrolysis at a faster or slower rate. It is a significant open question whether such systematic bias can be corrected for or avoided, for example, by appropriate strand design and experimental protocols.

In conclusion we have shown that it is theoretically possible to reduce errors due to false negative, false positives, and lost strands to tolerable levels in any extract-based DNA computation with only small extra time and space factors. Although these techniques should be useful immediately for larger problems than have been demonstrated to date, it is clear that more significant applications of DNA computing will require substantial improvements in the underlying biotechnology.

References

[1] Leonard M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266:1021–1024, November 11, 1994.

[2] Richard J. Lipton. DNA solutions of hard computational problems. *Science*, 268:542–544, 1995.

[3] Dan Boneh, Christopher Dunworth, and Richard J. Lipton. Breaking DES using a molecular computer. In Lipton and Baum [**30**], pages 37–65.

Notation used in this paper.

symbol	interpretation
q	$Prob$(strand survives an extract step)
$\epsilon = 1 - p$	$Prob$(false negative or false positive | strand survives the step)
P_s	$Prob$(strand survives entire computation)
$P_{survival}$	$Prob$(at least one of R copies survives the computation)
P_c	$Prob$(strand is correctly processed by a compound extract)
P_e	$= 1 - P_c$
$P_{correct}$	$Prob$(strand is correct after the computation | strand survives)
$k = \log_q \frac{1}{2}$	the number of extract steps per duplicate step
E	the number of layers in the original algorithm
R	the number of copies of each strand initially (the redundancy)
n	the number of phases (layers) in a compound extract
δ	target error rate per step
m	the number of generations of "lose-half-then-duplicate"
Z_m	the number of strands in generation m
$P_{m,i}$	$Prob$(generation m has exactly i strands)
$G_m(s)$	$= \sum_{j\geq 0} P_{m,j} s^j$, the generating function for $P_{m,i}$
$Q_m = 1 - P_{m,0}$	$Prob$(at least one strand survives to generation m)
b	for SAT problems, the number of binary variables in the input
M_i	set of inputs that can reach the yes-tube only after i errors
N_{bad}	the number of bad strands in the final yes-tube

TABLE 2

[4] Paul W. K. Rothemund. A DNA and restriction enzyme implementation of Turing Machines. In Lipton and Baum [**30**], pages 75–119.
[5] Warren D. Smith. DNA computers in vitro and in vivo. In Lipton and Baum [**30**], pages 121–185.
[6] Donald Beaver. A universal molecular computer. In Lipton and Baum [**30**], pages 29–36.
[7] Erik Winfree. On the computational power of DNA annealing and ligation. In Lipton and Baum [**30**], pages 199–221.
[8] Erik Winfree, Xiaoping Yang, and Nadrian C. Seeman. Universal computation via self-assembly of DNA: Some theory and experiments. In Landweber and Baum [**31**].
[9] Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman. Design and self-assembly of two-dimensional DNA crystals. *Nature*, 394:539–544, 1998.
[10] Qi Ouyang, Peter Kaplan, Shumao Liu, and Albert Libchaber. DNA solution of the maximal clique problem. *Science*, 278:446–449, 1997.
[11] Qinghua Liu, Anthony G. Frutos, Liman Wang, Anne E. Condon, Robert M. Corn, and Lloyd M. Smith. DNA computating on surfaces. In Winfree and Gifford [**32**].
[12] Hiroshi Yoshida and Akira Suyama. Solution to 3-SAT by breadth first search. In Winfree and Gifford [**32**].
[13] Dirk Faulhammer, Anthony R. Cukras, Richard J. Lipton, and Laura F. Landweber. When the knight falls: On constructing an RNA computer. In Winfree and Gifford [**32**].
[14] Richard M. Karp, Claire Kenyon, and Orli Waarts. Error-resilient DNA computation. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 458–467, Providence, RI, January28–30 1996. AMS/SIAM.
[15] Sam Roweis and Erik Winfree. On the reduction of errors in DNA computation. *Journal of Computational Biology*, 6(1):65–75, 1999.
[16] Eric B. Baum. DNA sequences useful for computation. In Landweber and Baum [**31**], pages 235–241.
[17] Kalim U. Mir. A restricted genetic alphabet for DNA computing. In Landweber and Baum [**31**], pages 243–246.

[18] R. Deaton, R. C. Murphy, M. Garzon, D. R. Franceschetti, and S. E. Stevens, Jr. Good encodings for DNA-based solutions to combinatorial problems. In Landweber and Baum [**31**], pages 247–258.

[19] R. Deaton, M. Garzon, R. C. Murphy, D. R. Franceschetti, and S. E. Stevens, Jr. Reliability and efficiency of a DNA-based computation. *Physical Review Letters*, 80(2):417–420, 1998.

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

[21] 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.

[22] Julia Khodor and David K. Gifford. The efficiency of sequence-specific separation of DNA mixtures for biological computing. In Rubin and Wood [**33**], pages 39–46.

[23] Junghuei Chen and David Harlan Wood. A new DNA separation technique with a low error rate. In Rubin and Wood [**33**], pages 47–56.

[24] Sam Roweis, Erik Winfree, Richard Burgoyne, Nickolas V. Chelyapov, Myron F. Goodman, Paul W. K. Rothemund, and Leonard M. Adleman. A sticker based architecture for DNA computation. In Landweber and Baum [**31**], pages 1–29.

[25] Dan Boneh, Christopher Dunworth, Richard J. Lipton, and Jiří Sgall. Making DNA computers error resistant. In Landweber and Baum [**31**], pages 163–170.

[26] Martyn Amos, Alan Gibbons, and David Hodgson. Error-resistant implementation of DNA computations. In Landweber and Baum [**31**], pages 151–161.

[27] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. John Wiley & Sons, 3 edition, 1968.

[28] Leonard M. Adleman, Paul W. K. Rothemund, Sam Roweis, and Erik Winfree. On applying molecular computation to the data encryption standard. *Journal of Computational Biology*, 6(1):53–63, 1999.

[29] Leonard M. Adleman. On constructing a molecular computer. In Lipton and Baum [**30**], pages 1–21.

[30] Richard J. Lipton and Eric B. Baum, editors. *DNA Based Computers: DIMACS workshop, April 4, 1995*, volume 27, Providence, RI, 1996. American Mathematical Society.

[31] Laura F. Landweber and Eric B. Baum, editors. *DNA Based Computers II: DIMACS Workshop, June 10-12, 1996*, volume 44, Providence, RI, 1998. American Mathematical Society.

[32] Erik Winfree and David K. Gifford, editors. *Proceedings of the* 5^{th} *DIMACS Meeting on DNA Based Computers, held at MIT , June 14-15, 1999*, preliminary, 1999.

[33] Harvey Rubin and David Harlan Wood, editors. *DNA Based Computers III: DIMACS Workshop, June 23-25, 1997*, volume 48 of *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.*, Providence, RI, 1999. American Mathematical Society.

(K. Chen) Department of Computer Science, Princeton University, Princeton, NJ 08544,

E-mail address: kcchen@phoenix.princeton.edu

(E. Winfree) Department of Molecular Biology, Princeton University, Princeton, NJ 08544, and, Departments of Computer Science and Computation and Neural Systems, California Institute of Technology, Pasadena, CA 91125

E-mail address: winfree@hope.caltech.edu