Making DNA computers error resistant

Dan Boneh
dabo@cs.princeton.edu

Christopher Dunworth
ctd@cs.princeton.edu

Richard J. Lipton*
rjl@cs.princeton.edu

Department of Computer Science
Princeton University
Princeton, NJ 08544

Jiří Sgall†

sgallj@mbox.cesnet.cz
Mathematical Institute, AV ČR
Žitná 25
115 67 Praha 1
Czech Republic

Abstract

We describe methods for making volume decreasing algorithms more resistant to certain types of errors. Such error recovery techniques are crucial if DNA computers ever become practical. Our first approach relies on applying PCR at various stages of the computation. We analyze its performance and show that it increases the survival-probability of various strands to acceptable proportions. Our second approach relies on changing the method by which information is encoded on DNA strands. This encoding is likely to reduce false negative errors during the bead separation procedure.

1 Introduction

In the short history of DNA (deoxyribonucleic acid) based computing there have already been a number of exciting results. It all started with Adleman's [A] beautiful insight that showed that biological experiments could solve the Directed Hamiltonian Path problem (DHP). Then, Lipton [L] showed how to use DNA to solve more general problems, namely to find satisfying assignments for arbitrary (directed) contact networks, which includes the important case of arbitrary formulas.

Since then there has been a series of papers on DNA computation [BDL,BDLS,S,B,RW,R,P]. Each of these subsequent results is of the following form: Given enough strands of DNA and certain

*Supported in part by NSF CCR–9304718.

†Partially supported by grants A119107 and A1019602 of AV ČR. Part of this work was done at Institute of Computer Science, Hebrew University, Jerusalem, Israel; partially supported by Golda Meir Postgraduate Fellowship.

biological operations; one can simulate some classic model of computation efficiently. Some compare to formulas, some to circuits, others to 1-tape Nondeterministic Turing Machines.

All results assume that the biological operations are *perfect*. Clearly, this is *not* the case and errors must be addressed. However, it is important to know what the ultimate limits are without errors. Clearly, if in the error-free case DNA cannot do exciting things, then there is no hope for the realistic case.

However, DNA operations are *not* error free. Our main result is that it is possible to allow a large class of DNA computations to survive fairly high error rates. This class of DNA computations includes Adleman's original algorithm as well as Lipton's formula SAT algorithm.

All DNA algorithms use strands of DNA to encode many parallel computations. They naturally fall into three classes: Algorithms are *Decreasing Volume* if the number of strands decreases as the algorithm executes; they are *Constant Volume* if all the strands remain throughout the computation. A *Mixed Algorithm* is one that fits into neither class. Adleman's and Lipton's are decreasing volume. Note, we use "volume" to denote number of strands.

Adleman's and Lipton's algorithms are even more special. In these algorithms each strand is either *good* or *bad*. A good strand encodes a solution to the problem; a bad strand does not. The goal of the algorithms is simple: to keep the good strands around and to eliminate the bad strands. If they succeed, then the final test tube will only contain good strands. The answer can then be found by sequencing one of the remaining strands.

The difficulty with this simple description is that errors can take place. An error may have terrible consequences. If a good strand is damaged or is lost, then the algorithm will fail. If on the other hand, bad strands are not removed, there may be many bad strands left at the end of the algorithm. This also could cause the algorithm to fail.

Errors come from two sources. First, every operation can cause an error. The principal operation in these algorithms is *extraction* [A,L]. This operation removes from a test tube those strands that contain some given pattern. However, extractions are not perfect. They usually only remove about 95% of the strands that match the pattern. In additional, strands that do not match will sometimes be removed anyway: rates are typically 1 part in 10^6. Another source of errors are due to [L2,SA] the fact that DNA has a "half-life": it decays at a finite rate. If the DNA algorithm takes months, for example, there is a probability that the good strands will simply dissolve.

The reason that errors in operations are so critical is that we envision algorithms that do thousands of steps. If each extraction removes 99% of the good strands, then the probability that a good strand is there at the end of a 1,000 step computation is tiny, i.e. is 0.00004. Thus, even at this high extraction rate the algorithm essentially will always fail. This does not even take into account the extraction of bad strands or the half-life problem.

Our first main result is the following: *We can show how to map Adleman's and Lipton's algorithms into a new algorithms that are constant volume. These new algorithms are highly resistant to errors.* Moreover, these new algorithms run in almost the same number of steps. Thus, the time penalty for making the algorithms error resistant is small.

However, there is a "space" penalty for making the algorithms constant volume. The number of test tubes remains the same throughout the computation. Thus, the same amount of equipment is needed at the beginning of the computation as later on. This may not be much of a penalty. If one only wishes to perform one computation, then the original algorithm would have more and more equipment sit idle as the volume decreases.

Our second result has to do with decreasing the error rates in the bead separation procedure. Lipton [L] describes how binary strings can be encoded as DNA strands in a useful manner. We suggest a modified encoding scheme which is likely to reduce the false negative error rate. We mention that recently Karp, Kenyon and Waarts [KKW] have also presented some techniques for coping with errors in the bead separation procedure.

2 Error resistant computations

We now turn to showing how to make DNA algorithms more error resistant. In order to make the analysis clear we will assume that the algorithm operates as follows: (The more general case in [L] is handled in the same way.) Initially, there are 2^n strands of DNA. Just one of these strands is good; all the rest are bad. Clearly, the case of *one* good strand is worst case.

The algorithm consists of s extraction steps. The good strand always matches the pattern of the extraction. However, a bad strand may or may not match the pattern. A *type I* (or *false negative*) error occurs when a strand is not correctly extracted: let p be the probability that this happens. A *type II* (or *false positive*) error occurs when a strand should not be extracted but is anyway: let q be the probability that this happens. Typical values of p are .95 and q are 10^{-6} (see [A]).

Note, just because a strand is bad does *not* mean that at each step it will not match the pattern. The good strand always matches the pattern; the bad strands may or may not. Indeed if x is a bad strand let $M(x)$ be the number of extractions for which it does *not* match the pattern. Clearly, $M(x)$ is at least 1.

We next assume that the computation decreases its volume at a uniform rate. Thus, we will assume from now on that every step decreases the volume at the same rate. Thus, since there are s steps and 2^n strands, the volume goes down by a factor of 2 every s/n steps. If $s = 900$ and $n = 60$, then every 15 steps the volume of DNA halves.

In the case that each step decreases the volume at the same rate, our analysis is simple. However, our method can easily be adjusted for a rate of decrease that is only approximately constant. Our method does not work in the pathological case where the volume goes down in some very complex way. We believe that it may be possible to randomize the order of extractions to "average" out the rate. Finally, such pathological cases may not arise in real problems.

Our modification to the basic algorithms is twofold: First, *every s/n steps double the amount of DNA.* We do this by a PCR operation [PCR]. There are several points: In practice one might wait until the volume decreased by a larger factor. We will do the analysis in this simple case; the generalization to bigger steps is quite easy. Second, the additional cost in time is small. Even if the PCR step is slow, it only happens every s/n steps. In our example, this is every 15 steps. Thus, the additional required time is a small percent of the total time.

Our second modification is to the final detect step. Before we assumed that ideally only one strand remained in the final test tube. Thus, to determine the answer we only had to sequence that strand. Now there may be both good and bad strands in the final test tube. We, therefore, replace this simple detect step by the following: Remove a strand from the test tube and sequence it. Then, check whether or not it is a solution. If it is, the algorithm has found a solution. If it is not, then try another strand. After some fixed number of these steps, say m, stop and report failure. Note, it is easy to computationally check whether or not a sequence does indeed encode a solution to the problem being solved.

The modified algorithm can now fail in two different ways. First, there might be no good strands in the final test tube. Second, there might be good strands in the final test tube, but there might be so many bad strands that the probability that the final step gets a solution is small. This leads naturally to the following: Let P_s be the *survival probability* i.e. the probability that some good strand is in the final test tube; also let P_r be the *selection probability* i.e. the probability that at least δ of the strands in the final test tube are good.

The probabilities P_s and P_r together control the probability that our algorithm succeeds. More precisely, we can use them to easily compute an upper bound on the probability that our algorithm fails. The probability of failure is at most

$$1 - P_s + P_s(1 - P_r) + P_s P_r (1 - \delta)^m$$

where recall that we do m repeated detects and δ is the amount of good strands in the final test tube. Thus, the probability that our method works is at least

$$P_s P_r (1 - \delta)^m.$$

The key is to bound these two probabilities, P_s and P_r. Let us first consider P_s. It should be clear that without the PCR step the probability that the good strand survives is p^s: recall that s extractions are performed and each succeeds with probability p. Now, every s/n steps the surviving strands are doubled in number. This applies equally to the good and the bad strands. Therefore, if a good strand survives s/n steps, it creates 2 new good strands. (Note, we are assuming that PCR step is error-free. However, any error there can be handled by decreasing p.)

This is an example of a *branching process* [F]. Branching processes are famous for modelling nuclear reactions and spread of diseases. In our case we are modelling whether or not the single initial good strand or its descendants survive many extractions. An example of such a branching process can be seen in Figure 1. Let r be the probability that a single good strand survives to the next PCR step, i.e. $r = p^{s/n}$. If a strand survives, then it becomes 2 good ones. Thus, the chances have greatly increased that some good strand will survive all the future extractions. Following Feller [F] the key quantity is the extinction probability ζ; for this branching process it is the smallest positive solution to the equation

$$1 - r + r\zeta^2 = \zeta.$$

A simple computation shows that the value of ζ is $\frac{1}{r} - 1$. The probability, P_s, that some good strand survives is $1 - \zeta$. Therefore,

$$P_s = 1 - \zeta = 2 - \frac{1}{r} = 2 - p^{-s/n}$$

Table 1 contains some example results. The probability that the original algorithms without PCR improvement succeed is essentially 0 in all cases. The table shows the probability that *some* good strands survive the entire series of steps.

It remains to show that P_r is large, i.e. that it is likely that the ratio of good strands to bad strands is not too small. For this argument we can *ignore* the PCR modification. The reason

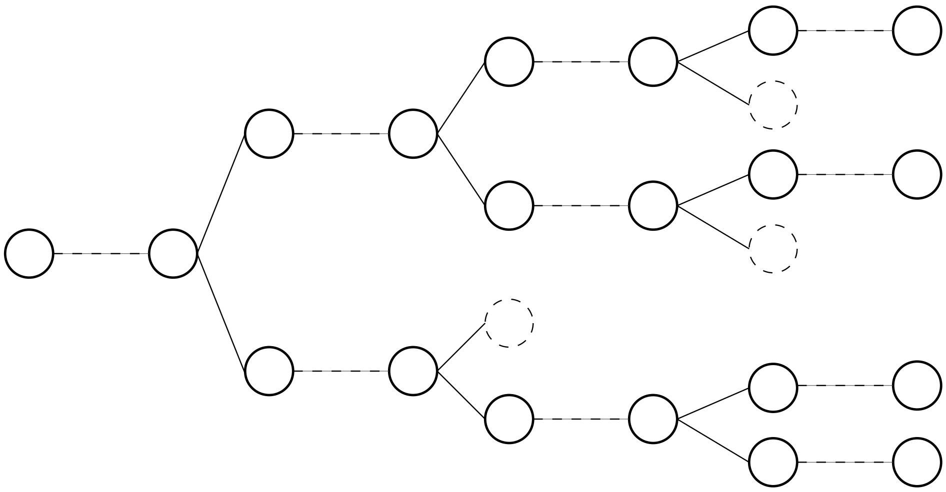

Figure 1: An example of a branching process. A good strand is replicated every few steps. Even though some of its descendants die (dashed discs), its prevalence increases exponentially.

is simple: we are interested only in the *ratio* of the number of good to bad strands. Since PCR doubles both the good and bad strands equally their ratio is unchanged. Thus, in this case we can assume that there is exactly 1 good strand.

Recall that a bad strand x does not match the pattern of an extract for $M(x)$ extractions. It is therefore the case that a bad x will survive only if $M(x)$ type II errors occur. Thus, the expected number of bad strands that survive is at most

$$\sum_x q^{M(x)}$$

where the sum is over all bad strands since q is the probability that a type II error occurs. (The actual sum is smaller since bad strands still might be subject to type I errors.)

In order to simply this sum define M_k as the number of bad strands x that have $M(x) = k$. Then, the above is equal to

$$M_1 q + M_2 q^2 + M_3 q^3 + \ldots$$

where the missing terms are insignificant. This follows since $q = 10^{-6}$ and there are at most 2^{70} strands.

The critical point is that $M_1 q + M_2 q^2 + M_3 q^3$ determines how long the final detect step takes. Let this quantity be l. Then, the detect step will likely require order l steps. Clearly, we want this

p	probability good survives
.97	.42
.98	.65
.99	.84

Table 1: $s = 900$ and $n = 60$

to be as small as possible. However, with out any assumption it *is* possible that l is large. Perhaps $M(x) = 1$ for most bad strands.

There are three points. First, in practical problems it seems that it would be rare for many bad strands to satisfy all but a few of the constraints. Typically one would expect that a bad strand might have $M(x)$ fairly large. Clearly, in this case l would be small. Note, even if $M_1 = 10^8$ and $M_2 = 10^{14}$ and $M_3 = 10^{20}$, the value of l is only 300.

Second, we can restructure the computation so that l is made smaller. For example, if l is large we can just repeat the whole computation and reduce l. If one repeats extractions twice, then all values of $M(x)$ are doubled. This is quite expensive and it would be better to avoid this. However, it is interesting that there is a general method to reduce l. Note, we do not have to do this repeating until we fail at the detect step.

Finally, it may be possible to do a more interesting transform on problems that will reduce the number of "almost" solutions, i.e. will reduce l. This seems to be related to the famous Probabilistic Checkable Proofs of [Ar].

The techniques presented above transformed a decreasing volume algorithm into a constant volume algorithm. By doing so we made sure that the prevalence of good strands increases in the test tube. However, this technique can not be applied to an algorithm which is constant volume to begin with. The reason is that the amount of material used by the algorithm would double after every PCR step. Within a few steps the amount of material would be to large to handle.

3 Double encoding of data

We propose a technique which is likely to to reduce the rate of false negative errors in the bead separation step. Our approach will hopefully increase the yield of the extraction procedure.

The basic idea is to modify the way in which binary strings are encoded in DNA. The hope is that the extraction procedure will incur less false negatives when this encoding is used. In Lipton's algorithm for solving SAT [L] a binary string $x = x_1 \dots x_n$ is represented by the DNA molecule

$$3' - B(1, x_1)\ B(2, x_2)\ B(3, x_3)\ \cdots\ B(n, x_n) - 5'$$

where $B(k, b)$ are 20-mers appropriately chosen. We refer to this method of encoding binary strings as the *standard encoding scheme*.

The traditional approach for coping with the false negative problem is to introduce many DNA strands encoding the same binary string into the solution. When there are "many" copies of the same DNA strand with "high" probability one of them will survive an extraction step despite the effect of false negative errors.

3.1 A modified encoding scheme

We propose to encode the binary string $x = x_1 \dots x_n$ using the following DNA molecule:

$$3' - B(1, x_1)\ B(2, x_2)\ \cdots\ B(n, x_n)\ B(1, x_1)\ B(2, x_2)\ \cdots\ B(n, x_n) - 5'$$

That is, each 20-mer $B(i, x_i)$ appears twice in the DNA strand in separate positions on the strand. Suppose we need to extract all strands in the solution that contain a specific sequence, say $B(1, 1)$.

	T_1	T_2
Type of encoding	Standard encoding	Double encoding
Type of DNA strands	D_1	D_2
Number of copies	$2r$	r
Extraction probability	p_1	p_2
survival prob. after k extractions	$S_1 = 1 - (1 - p_1^k)^{2r}$	$S_2 = 1 - (1 - p_2^k)^r$

Table 2: Comparison of standard and double encoding schemes

The hope is that since $B(1,1)$ appears twice on the DNA strand, that DNA strand is more likely to be extracted than a DNA strand containing only one copy of $B(1,1)$. Theoretically, this seems to be the case since it becomes easier for the strand to attach itself to some magnetic bead.

We refer to this modified method of encoding binary string as *double encoding.* Note that since the DNA strands in the solution are now twice as long, the solution can contain only half the number of strands. As a result only half as many strands can represent a given binary string. It is therefore unclear that using double encoding makes the algorithm more resistant to false negative errors.

A simple analysis can be used to show that double encoding is likely to make the SAT algorithm of Lipton more resistant to false negative errors. Let D_1 be a DNA strand containing the sequence B once. Let D_2 be a DNA strand containing the sequence B twice in separate locations on the strand. Define p_1 (resp. p_2) to be the probability that D_1 (resp. D_2) is extracted during an extraction of all strands in the solution containing the sequence B. We are conjecturing that $p_2 > p_1$ though this still has to be verified by experiments.

Let T_1 be a solution containing $2r$ copies of D_1. Similarly, let T_2 be a solution containing r copies of D_2. Now suppose we perform k extract operations on each of T_1 and T_2. It can easily be seen that the probability that at least one of the strands in T_1 survives the k extractions is

$$S_1 = 1 - (1 - p_1^k)^{2r} \quad .$$

Similarly, the probability that at least one of the strands in T_2 survives the k extractions is

$$S_2 = 1 - (1 - p_2^k)^r \quad .$$

The above data is summarized in Table 2. The question we need to answer is: how large should p_2 be so that $S_2 > S_1$. Some simple algebra shows that $S_2 > S_1$ when $p_2 > p_1(2 - p_1^k)^{1/k}$. Table 3 contains some values for this threshold quantity.

For example, suppose we ran an algorithm which requires 10 extraction steps. That the above table shows that if p_2 is bigger than p_1 by 4.6% then the double encoding method will do better than the standard encoding. Note that the correct value of p_2 can only be determined through experimentation.

3.2 Constructing the initial solution

The first steps in the SAT algorithm is the construction of DNA strands which represent all n bit binary strings. For the standard encoding this is done by creating all paths in a certain graph as is described in [L].

The generation of the initial solution for the double encoding is a bit more complicated. We can no longer simply form all paths in some graph. Fortunately we can use a technique described in [BDLS]. In [BDLS] it was shown that it is possible to create a solution of all pairs $(x, f(x))$ for a function f. By taking f to be the identity function we can create the required double encoding solution which consists of all strings (x, x).

4 Conclusions

We discussed methods for making the formula satisfaction techniques of Lipton [L] resistant to certain types of errors by applying PCR every several steps. This doubling step turns this algorithm into a constant volume algorithm. The analysis demonstrates that the error parameters of the extract operation are sufficiently small to make the formula satisfaction algorithm work in the presence of errors.

We also presented a modified encoding scheme (double encoding) which may be helpful in reducing error rates that arise in the bead separation procedure. The usefulness of this technique is to be determined through experimentation.

References

[A] L. Adleman, "Molecular Computation of Solutions to Combinatorial Problems", Science 266:1021-1024 (Nov. 11) 1994.

[Ar] S. Arora, "Probabilistic Checkable Proofs", Ph.D. thesis, U.C. Berekely, 1994.

[B] D. Beaver, "A Universal Molecular Computer", Penn State University Tech Report CSE-95-001.

[BDL] D. Boneh, C. Dunworth, R. Lipton, "Breaking DES Using a Molecular Computer", Princeton CS Tech-Report CS-TR-489-95.

[BDLS] D. Boneh, C. Dunworth, R. Lipton, J. Sgall, "On the Computational Power of DNA", Princeton CS Tech-Report CS-TR-499-95.

[PCR] H. Erlich, "PCR Technology: Principles and Applications for DNA Amplification", Stockton Press.

k	$p_{2\,\min} = p_1(2 - p_1^k)^{1/k}$
5	96.4%
10	94.6%
15	93.6%

Table 3: Threshold value for p_2 assuming $p_1 = 90\%$

[F] W. Feller, "An Introduction to Probability Theory and Its Application", Volume I., John Wiley and Sons 1957.

[KKW] R. Karp, C. Kenyon, O. Waarts, "Error-resilient DNA computations", In Proceedings of Symposium on Discrete Algorithms, SODA 1996.

[L2] Lindahl, Tomas, "Instability and decay of the primary structure of DNA", Nature 362 (Apr 1993) 709-715

[L] R. Lipton, "Using DNA to solve NP-Complete Problems", Science 268:542-545 (Apr. 28) 1995.

[P] C. Papadimitrou, private communications.

[RW] D. Rooßand Klaus W. Wagner, "On the power of Bio-Computers", Technical report, University of Würzburg, 1995.

[R] Rothemund, A DNA and restriction enzyme implementation of Turing machines, unpublished manuscript.

[SA] Saul, R.L., Ames, B.N., "Background levels of DNA damage in the population", Basic Life Sciences 38 (1986) 529-535.

[S] W. Smith, "DNA Computers in Vitro and Vivo", Technical report, NEC, 1995.