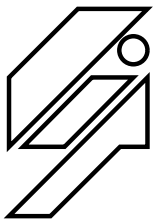
Error-resilient DNA computation.

Richard M. Karp, Claire Kenyon, Orli Waarts



Laboratoire de l'Informatique du Parallélisme

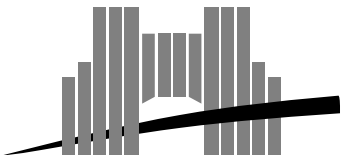
Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Error-Resilient DNA Computation

Richard M. Karp
Claire Kenyon
Orli Waarts

September 1995

Research Report N^{o} 95-20



Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Error-Resilient DNA Computation

Richard M. Karp
Claire Kenyon
Orli Waarts

September 1995

Abstract

The DNA model of computation, with test tubes of DNA molecules encoding bit sequences, is based on three primitives, Extract-A-Bit, Merge-Two-Tubes and Detect-Emptiness. Perfect operations can test the satisfiability of any boolean formula in linear time. However, in reality the Extract operation is faulty. We determine the minimum number of faulty Extract operations required to simulate a single highly reliable Extract operation, and derive a method for converting any algorithm based on error-free operations to an error-resilient one.

Keywords: Reliability, DNA Computations, Lower Bounds, Algorithms

Résumé

Le modèle de calcul basé sur les molécules d'ADN codant des suites de bits utilise trois primitives, Extraction d'un bit, Fusion de deux éprouvettes, et Détection d'éprouvette vide. Avec des opérations fiables, on peut tester la satisfiabilité d'une formule booléenne quelconque en temps linéaire. Mais en réalité, l'opération d'Extraction est peu fiable. Nous déterminons le nombre minimum d'Extractions non fiables et de fusions permettant de simuler une Extraction très fiable, puis donnons une réduction qui, d'un algorithme basé sur des primitives parfaitement fiables, déduit un algorithme lorsque l'opération Extraction est non fiable.

Mots-clés: Fiabilité. Calculs avec l'ADN, Bornes Inférieures, Algorithmes

Error-Resilient DNA Computation

Richard M. Karp* Claire Kenyon† Orli Waarts‡

1 Introduction

Lipton [12], building upon the earlier work of Adleman [1], has proposed a model of computation using DNA molecules. In this model each operation is performed on a *test tube*, *i.e.*, a set of DNA strands, each of which encodes a sequence $x = (x_1, x_2, \ldots, x_n)$ of bits. The *Extract* operation on bit x_i divides a test tube into two test tubes, one containing the strands with $x_i = 1$, and the other containing the strands with $x_i = 0$. The *Merge* operation forms the union of two test tubes, and the *Detect* operation tests whether a test tube is empty. Several additional operations have been considered, including *Duplicate*, which makes two copies of a test tube, and *Append*, which appends the same bit to each strand in a test tube.

Assuming the Extract, Merge, and Detect primitives are perfect, they can be used to test the satisfiability of any boolean formula in a number of operations proportional to its size. These primitives, together with the Append operation, can be used to evaluate any boolean circuit in a number of operations proportional to its size. In the short time since Adleman's original paper appeared there have been a multitude of further papers showing how these and other operations may be used to solve various classes of decision and optimization problems (cf. [1, 2, 3, 4, 5, 6, 7, 12, 18]). Except for [7], which we discuss below, the correctness of all these solutions depends on all molecular

*University of California at Berkeley and ICSI. E-Mail: karp@cs.berkeley.edu

†LIP, URA CNRS 1398, ENS-Lyon. Part of this research was performed while this author was at the University of California at Berkeley. E-Mail: Claire.Kenyon@lip.ens-lyon.fr

‡University of California at Berkeley. Work supported in part by NSF postdoctoral fellowship. E-Mail: waarts@cs.berkeley.edu

biology experiments working perfectly without any errors. However, clearly, as pointed out by Adleman [1], the Extract operation is error-prone. During each Extract, each strand involved has some chance of ending up in the wrong test tube. The Merge primitive is simpler to implement, and can be assumed to be error-free. As we argue below, the Detect primitive is not relevant for our procedures, and hence we are not concerned with its reliability in this paper.

The error in the Extract operation is crucial: it will most likely destroy any computation that tries to ignore it. For example, a widely used encryption procedure is the Data Encryption Standard or DES [13]. [5] presents a molecular program for implementing a chosen plaintext attack on DES. Given a 64-bit plaintext, the program evaluates the DES circuit concurrently for all possible 56-bit keys. The program proceeds in 916 steps, of which about half are error-prone Extract steps (the number of steps can be reduced to 664 using an additional operation called Join, which may not be realizable experimentally). Typically, the probability that a one bit will be misclassified as a zero in an Extract operation is approximately .05, and the probability that a zero bit will be misclassified as a one is approximately 10^{-6}. Thus, with very high probability, after the 916 steps, and at least four months of computations [5], the probability that a correct output will be obtained for any given key will be minuscule.

Thus, it seems that any hope that DNA computation will ever be practical depends on finding efficient general transformations to make DNA algorithms error-resilient, as well as on studying the inherent limitations of efficiency of such transformations. Both the transformations and the lower bounds should of course take into account the fact that each operation can operate on a huge number of objects simultaneously. This property is one of the main issues that distinguish DNA computers from conventional computers. Not surprisingly, it also distinguishes the problem of making DNA computations error-resilient from the issue of computing with unreliable operations on conventional computers, that has been the focus of much research (cf. [8, 9, 10, 14, 15, 16, 17]).

In this paper we provide a method for making computations error-resilient without a big sacrifice in their running time. Moreover, we derive lower bounds on the cost of such methods.

We start with the problem that is at the core of all error-resilient DNA computations: simulate a highly reliable Extract operation. Specifically,

given a set of strands T, each containing the encoding of a single bit, separate T into two sets, $+T, -T$, so that each bit that is a one will end up in $+T$ with probability $\geq 1-\delta_1$, and each bit that is a zero will end up in $-T$ with probability $\geq 1-\delta_2$. We seek a sequence of operations that achieves this result no matter how many bits are initially present. Under this requirement the Detect operation is not useful, since no test tube will be empty if the number of initial strands is sufficiently high. Thus our program will consist of a fixed sequence of Extract and Merge operations. Refer to this problem as the *error-resilient bit evaluation* problem. Let ϵ be the probability that a single Extract will misclassify a one bit as a zero, and let γ be the probability that a single Extract will misclassify a zero bit as a one. All errors are assumed to be independent.

In any procedure for the error-resilient bit evaluation problem based on Extracts and Merges, the number of Merges is bounded by the number of Extracts. Thus we take the number of Extracts as our measure of complexity. Our main result is that the inherent complexity of the error-resilient bit evaluation problem is $\Theta(\lceil \log_{\frac{1-\gamma}{\epsilon}} \frac{1}{\delta_1} \rceil \cdot \lceil \log_{\frac{1-\epsilon}{\gamma}} \frac{1}{\delta_2} \rceil)$. The upper bound is constructive. The lower bound is the principal technical achievement of the paper. Its difficulty arises already in the case where $\delta_1 = \delta_2$ and $\gamma = \epsilon$. Its derivation is based on the analysis of a novel potential function.

One step up in granularity from bit evaluation is the problem of evaluating a boolean function. In this problem one is given a set of strands, each representing a sequence of n bits. Refer to the ith bit of strand x by x_i. Given a boolean function $f(x_1, x_2, \ldots, x_n)$, one wishes to create two test tubes, one containing those strands x for which $f(x) = 1$, and the other containing those strands x for which $f(x) = 0$. Error resiliency will require that each strand will end up in the wrong tube with probability not greater than some δ. We refer to this problem as the *weak error-resilient function evaluation* problem. We also define a stronger and more desirable requirement in which the probability of a strand ending up in the wrong tube may depend on the type of the strand. Intuitively, if there are many wrong strands of a given type, we would like each of them to have only a small probability of error, since otherwise their number in the wrong tube will overwhelm the number of correct strands that ended up there. We refer to this variant as the *strong error-resilient function evaluation* problem or just the *error-resilient function evaluation* problem.

We show that our error-resilient bit evaluation can be used modularly and efficiently to transform any DNA algorithm for evaluating some function into an error-resilient algorithm for evaluating the same function. The efficiency of the transformation does not depend on the number of strands but only on their sizes, on the required level of confidence, and on the errors in a single Extract. In particular we show that, for any function that can be evaluated using $O(a(n))$ perfect Extracts we can get an error-resilient algorithm that uses only $O(a(n) \cdot \lceil \log_\epsilon \frac{\delta}{a(n)} \rceil \cdot \lceil \log_\gamma \frac{\delta}{a(n)} \rceil)$ Extracts. We then show that for several n-variable functions, including Parity, Disjunction and Conjunction, no algorithm can do better than $\Omega(n \cdot \lceil \log_\epsilon \delta \rceil \cdot \lceil \log_\gamma \delta \rceil)$ Extracts. Our algorithm is a strong error-resilient function evaluation algorithm, while the lower bound applies even for weak error-resilient function evaluation. Notice that for $\delta = O(1/n)$, the upper and lower bounds for Parity, Conjunction and Disjunction match. In practice one would usually want $\delta \leq 1/n$, since the number of wrong strands is usually much greater than n times the number of correct strands, and hence if $\delta \neq O(1/n)$, the number of wrong strands in the tube that is intended to contain the correct strands, will overwhelm the number of correct strands there.

This is a preliminary version, with the proofs only sketched.

1.1 Other Related Work

The fact that the Extract operation is error-prone was first pointed out in [1]. Adleman estimated that ϵ, the probability for an Extract operation to misclassify a 1 bit, is about .1 and γ, the probability for an Extract to misclassify a 0 bit, is about 10^{-6}.

The papers [1] and [7] consider computations in which 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 in this class is to eliminate the bad strands, leaving only the good strands in the final tube. The authors focus on the case in which a computation is a sequence of tests, such that a strand is good if and only if it gives a positive result on each test. In the simplest case, where each test consists of an Extract on a single bit, Adleman's approach to making the computation error-resilient when ϵ is much larger than γ is based on trying each Extract several times, and collecting in a test tube all those strands that give a positive result in any of the trials. For example, if $\epsilon = .1$, $\gamma = 10^{-6}$ and the number of trials is 6, then the

probability of misclassifying a good strand is 10^{-6}, and the probability of misclassifying a bad strand is less than 6×10^{-6}. Adleman shows that, in a particular numerical example, an algorithm modified in this way has a good chance of creating a final test tube that contains no bad strands and contains a good strand unless all the strands in the initial test tube were bad.

In [7], Boneh and Lipton noted that even faulty tests will eliminate the bad strands at a faster rate than the good strands. They assume that, if the initial numbers of both types of strands are large, the ratio of good strands to bad strands will grow at a steady exponential rate in the course of the process, so that the good strands will eventually dominate. They note, however, that if the initial number of good strands is very small then the good strands may die out. To prevent this they suggest using the Duplicate primitive to double the total volume of remaining DNA whenever the volume drops below half of its original amount. (This is possible because the total volume of remaining DNA decreases in the course of the algorithm.) Their method effectively transforms the problem in which the initial volume of DNA is finite, into a problem in which it is arbitrarily large, and hence as they show, it yields a reasonably high probability that the good strands will not die out, and will occupy most of the volume of the final test tube.

Our error-resilient procedures are more powerful than the Adleman and Boneh-Lipton procedures. Our procedures ensure that every strand, whether good or bad, is classified correctly with the required probability. Thus they apply to tasks such as evaluating a boolean function or breaking DES, which do not conform to the paradigm of discarding the bad strands and keeping the good ones, and in addition, they can be incorporated into any algorithm in a modular fashion. Further, the Boneh-Lipton procedure requires the Duplicate primitive, whereas our procedures use only Extract and Merge. As mentioned by Adleman, the Duplicate operation is a concern: its cost and complexity is on a larger scale than Extract and Merge; it itself introduces errors; and it may not even be feasible in case the DNA medium is replaced by inorganic material, as is being hoped [1]. Thus, as concluded by Adleman [1], for the purposes of a practical molecular computer, it may be preferable to avoid it or restrict its use as much as possible.

2 The Model

A tube is a set of DNA strands, each encoding a sequence of bits $x = (x_1, x_2, \ldots, x_n)$. Given a tube, one can perform the following operations:

1. *Extract.* Given a tube T, produce two tubes $+T$ and $-T$ where $+T$ is all of the strands of T which contain 1 in the tested bit and $-T$ is all of the strands of T which do not contain 1 in this bit.

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

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

After an Extract, a strand which should end up in $+T$ ends up in $-T$ with probability ϵ, and a strand that should end up in $-T$ ends up in $+T$ with probability γ.

In this abstract we assume that $\epsilon + \gamma < 1$. The case where $\epsilon + \gamma > 1$ is reduced to this case by reversing the roles of the tubes $+T$ and $-T$. When $\epsilon + \gamma = 1$ the Extract operation treats one and zero bits alike; this case is both uninteresting in practice and trivial to analyze.

3 Error-Resilient Bit Evaluation

We are given an initial tube containing n strands, each of which consists of a single binary bit. The strands containing 0 are called 0-strands, and the strands containing 1 are called 1-strands. The goal is to separate the strands into two test tubes, called the 0-tube and the 1-tube, such that:

1. each 1-strand has probability at most δ_1 of ending up in the 0-tube; and

2. each 0-strand has probability at most δ_2 of ending up in the 1-tube.

Our procedure is required to satisfy this requirement regardless of the value of n. Under this requirement the Detect operation is of no value, since, when n is sufficiently large, there are many strands of both types and

every tube produced in the course of the procedure will be nonempty with high probability. Thus we may restrict attention to procedures consisting of Extract and Merge operations only.

4 Tight Bounds

In the full paper we show:

Theorem 4.1 *The number of Extract operations required for achieving error-resilient bit evaluation is* $\Theta\left(\lceil \log_\epsilon \delta_1 \rceil \cdot \lceil \log_\gamma \delta_2 \rceil\right)$.

Due to lack of space, in this abstract we will prove this theorem for $\delta_1 = \delta_2 = \delta$.

4.1 Intuition

The ideas behind our analysis of the error-resilient bit evaluation problem are clearest in the case where $\epsilon = \gamma$ and $\delta_1 = \delta_2 = \delta$. At any time during an algorithm, each strand has a *count*, defined to be equal to $i - j$ if the strand has been involved in $i + j$ Extracts, i of which classified the strand as a 1 and j of which classified the strand as a 0. Each strand's count behaves like a biased random walk on the count axis. The 1 strand is biased $(\epsilon, 1-\epsilon)$ and the 0 strand is biased $(1-\epsilon, \epsilon)$. Each strand starts its walk at the zero point. If the process starts with equally many 0-strands and 1-strands then, at any step, a random strand with count i has probability $\frac{\epsilon^i}{(1-\epsilon)^i + \epsilon^i}$ of being a 0-strand.

Roughly speaking, an algorithm can classify a strand as a 1 only when the strand's count is at least $\log_\epsilon \delta$, since then the strand's probability of being a 0 is at most δ, and hence the probability that a 0-strand will be misclassified is at most δ, as required. Therefore, each 1-strand needs to cross the right barrier of $\log_\epsilon \delta$ on the count axis before it can be classified as a 1. Similarly, a strand has to cross a left barrier of $-\log_\epsilon \delta$ on the count axis before being classified as a 0.

Since the number of strands in a tube is arbitrary, each time we perform an Extract on a tube, an arbitrary number of strands proceed in their random walk. Thus, we have an arbitrary number of random walks being performed in parallel.

The goal of the analysis is to determine the number of Extracts necessary and sufficient in order to have nearly all the strands cross their barriers.

The algorithm is pretty straightforward: We show that it is enough to proceed in $O(\log_\epsilon \delta)$ phases, in each of which one performs an Extract on each tube present, and then merges tubes that contain strands with identical counts. In this method we have exactly i tubes in the ith phase, and hence we immediately get an $O(\log_\epsilon^2 \delta)$ upper bound. Next, for $\epsilon \neq \gamma$, we replace each Extract by what we call a *Super Extract*. A Super Extract is a series of Extracts on the same bit, at the end of which each strand is in the wrong tube with probability $\leq \beta$. Now we can proceed with the algorithm for $\epsilon = \gamma = \beta$, using the Super Extract as a primitive step. The algorithm for the case $\delta_1 \neq \delta_2$ requires further ideas, and we defer its presentation to the full paper.

The lower bound requires a considerably deeper insight into the nature of this process and the possible interaction among the arbitrarily many random walks done in parallel. One difficulty is that one may merge tubes containing strands that are at different positions in their random walks. (Indeed our algorithm for the case $\epsilon \neq \gamma$ does so when performing a Super Extract step.)

For simplicity, assume $\epsilon = \gamma$. First consider an algorithm that never merges two tubes of unequal count. The key observation is that, at any point of the computation, for each point l on the count axis, a certain fraction return(l) of the good strands are expected to pass through l in the future. Part of this fraction consists of good strands whose random walk is currently at l or to its left, and the rest consists of the expected fraction of strands that will return to l from the right. Each time we perform an Extract on a tube with count l, at most a fraction return(l) of the good strands execute a step of their random walk, and among these a fraction $\frac{\epsilon}{1-\epsilon}$ are expected to return to l. Thus the step reduces return(l) by at most the factor $\frac{\epsilon}{1-\epsilon}$, and it turns out that, for all integer points $s \neq l$ on the count axis, return(s) remains unchanged. Also, using the fact that nearly all the good strands must have a count close to $\log_\epsilon \delta$ when the computation terminates, we obtain an upper bound $\alpha(l)$ on the value of return(l) at the end of the computation.

Since return(l) is initially equal to 1, is reduced by at most the factor $\frac{\epsilon}{1-\epsilon}$ when an Extract is performed at l, is not reduced at any other step, and must eventually be reduced below $\alpha(l)$, we find that the number of Extracts on tubes of count l must be at least $\log_{\frac{\epsilon}{1-\epsilon}} \alpha(l)$. Summing over all l in the

interval $[0..\log_\epsilon(\delta)]$, we obtain the lower bound of $\Omega(\log_\epsilon^2 \delta)$.

The proof that the lower bound holds even for algorithms that merge tubes of unequal counts is more difficult, and is achieved through the analysis of a novel potential function related to the function return(l).

4.2 The Lower Bound

4.2.1 Definitions

With each test tube T, we associate a *likelihood*, defined as

$$\frac{\Pr\{T(st) \mid st \text{ is a } 0\}}{\Pr\{T(st) \mid st \text{ is a } 1\}} ,$$

where st is a strand drawn at random from the initial test tube and $T(st)$ denotes the event that the strand st occurs in test tube T. Define the likelihood of a strand to be the likelihood of the tube which it belongs to. Next we define the *log-likelihood* of a test tube as

$$\log_{\frac{\gamma}{1-\epsilon}} \left(\frac{\Pr\{T(st) \mid st \text{ is a } 0\}}{\Pr\{T(st) \mid st \text{ is a } 1\}} \right) .$$

Define the log-likelihood of a strand to be the log-likelihood of the tube which it belongs to.

Clearly, initially all strands have likelihood 1, and hence log-likelihood 0.

We first focus on the strands that are 1's. We refer to these strands as *good*. At any time during the course of an algorithm, the log-likelihood of the good strands have a certain distribution μ. More precisely, $\mu(x)$ at time t is the expected fraction of good strands with log-likelihood $\leq x$ at time t.

Define

$$D = \log_{\frac{1-\epsilon}{\gamma}} \frac{1}{\delta} , \qquad \text{and} \qquad R = \log_{\frac{1-\gamma}{\epsilon}} \frac{1}{\delta} .$$

Define

$$\text{return}(l) = \int_{-\infty}^{l} d\mu(x) + \int_{l+}^{\infty} \left(\frac{\gamma}{1-\epsilon} \right)^{x-l} d\mu(x) .$$

Finally define the potential function associated to distribution μ as:

$$\Psi(\mu) = \int_0^D \log_{\frac{1-\gamma}{\epsilon}} \frac{\text{return}(l)}{\delta} dl .$$

4.2.2 Analysis

Initially, Ψ is $D \cdot R$. We will prove that an Extract decreases Ψ by at most 2, that a Merge cannot decrease Ψ, and that in the end Ψ is approximately $DR/2$. This will imply that the number of Extracts required for achieving error-resilient bit evaluation is approximately $\log_{\frac{1-\gamma}{\epsilon}} \frac{1}{\delta} \cdot \log_{\frac{1-\epsilon}{\gamma}} \frac{1}{\delta}/4$, and the lower bound will follow.

In the sequel, by a *fraction α of good strands in a tube*, we mean that the mass of good strands in this tube consitutes an α fraction of the total mass of good strands.

Claim 4.2 *Assume we do an Extract on a test tube T which contains an expected fraction α of good strands and has likelihood c. Then*

1. *(a) $+T$ contains expected fraction $(1-\epsilon)\alpha$ of good strands, and*
   (b) the likelihood of $+T$ is $\frac{\gamma}{1-\epsilon} \cdot c$;
2. *(a) $-T$ contains expected fraction $\epsilon\alpha$ of good strands, and*
   (b) the likelihood of $-T$ is $\frac{1-\gamma}{\epsilon} \cdot c$.

The proofs of the following two lemmas are given in the Appendix.

Lemma 4.3 *An Extract decreases Ψ by at most 2.*

Lemma 4.4 *A Merge can not decrease Ψ.*

Lemma 4.5 *When the algorithm stops, $\Psi \leq \frac{DR}{2} + D \cdot \log_{\frac{1-\gamma}{\epsilon}} 2 + \frac{R}{2}$.*

Proof: By definition of the error-resilient bit evaluation problem, a strand is correctly classified with probability $\geq 1-\delta$. Hence the fraction of good strands that are classified as good strands is some $p \geq 1-\delta$, and the fraction of bad strands that may be classified as good strands is some $q \leq \delta$. Thus, the average of the likelihood at the end of the algorithm of the good strands that are classified as good strands is $\frac{q}{p} \leq \frac{\delta}{1-\delta}$.

Let $\mu(x)$ be the fraction of good strands with log-likelihood $\leq x$ when the algorithm completes its Merges and Extracts. Let $S(x)$ be the set of all good strands that are eventually classified as good strands and that have log-likelihood $\leq x$ when the algorithm completes its Extracts and Merges.

Let R be the set of all good strands that are eventually classified as good strands. Define $\nu(x)$ as $S(x)/R$. Then, the average likelihood at the end of the algorithm of the good strands classified as good is $\int_{-\infty}^{\infty} \left(\frac{\gamma}{1-\epsilon}\right)^x d\nu(x)$. Thus, by the reasoning in the first paragraph we have:

$$\int_{-\infty}^{\infty} \left(\frac{\gamma}{1-\epsilon}\right)^x d\nu(x) = \frac{q}{p} \leq \frac{\delta}{1-\delta} \ .$$

The last equation implies that there is some r such that:

$$\int_{r+}^{\infty} (\frac{\gamma}{1-\epsilon})^x d\mu(x) \leq \delta \ , \qquad \text{and} \qquad \int_{r+}^{\infty} d\mu(x) \geq 1-\delta \ .$$

(In fact, one may need to take into account just a fraction of the mass at $r+$ (instead of the whole mass at $r+$) in order to satisfy the above two equations. To avoid introducing additional notation, imagine shifting an infinitesimal distance to the left, all mass that is at $r+$ but that shouldn't be taken into account in the above equations.)

Claim 4.6 $\quad return(0) \leq 2\delta$.

Proof: First note that by choice of r, for at most δ fraction of the good strands the eventual log-likelihood is $\leq r$. Thus, if $r \leq 0$, then the claim follows immediately from the fact reasoned above that $\int_{r+}^{\infty} (\frac{\gamma}{1-\epsilon})^x d\mu(x) \leq \delta$.

Assume $r > 0$. Let g be the fraction of the good strands whose eventual log-likelihood is in $(0, r]$, and let h be the fraction of good strands whose eventual log-likelihood is in $(\infty, 0]$. Then, clearly

$$\int_{0+}^{r} \left(\frac{\gamma}{1-\epsilon}\right)^x d\mu(x) \leq g \cdot \left(\frac{\gamma}{1-\epsilon}\right)^0 \leq g \ .$$

Thus,

$$\text{return}(0) = \int_{-\infty}^{0} d\mu(x) + \int_{0+}^{r} \left(\frac{\gamma}{1-\epsilon}\right)^x d\mu(x) + \int_{r+}^{\infty} \left(\frac{\gamma}{1-\epsilon}\right)^x \leq h + g + \delta \leq 2\delta \ .$$

The last inequality follows again from the fact that by the choice of r, at most fraction δ of the good strands have final log-likelihood $\leq r$, and hence $h + g \leq \delta$. ■

But the function return cannot increase very quickly. In fact, inspection of its definition shows that for all l, $\text{return}(l) \leq (\frac{1-\epsilon}{\gamma})^l \cdot \text{return}(0)$. Since $\text{return}(0) \leq 2\delta$, it follows that for all l, $\text{return}(l) \leq (\frac{1-\epsilon}{\gamma})^l \cdot 2\delta$.

Thus we get an upper bound on Ψ:

$$\Psi(\mu) \leq \int_0^D \log_{\frac{1-\gamma}{\epsilon}} \left(2\left(\frac{1-\epsilon}{\gamma}\right)^l\right) dl \leq D \cdot \log_{\frac{1-\gamma}{\epsilon}} 2 + \frac{DR}{2} + \frac{R}{2} .$$

(The algebraic manipulations are omitted from this extended abstract.) ■

Lemmas 4.3, 4.4, and 4.5, together with the fact that initially $\Psi = DR$, immediately imply:

Corollary 4.7 *The number of Extract operations required for achieving error-resilient bit evaluation is at least*

$$\frac{1}{4}\log_{\frac{1-\epsilon}{\gamma}}\frac{1}{\delta} \cdot \log_{\frac{1-\gamma}{\epsilon}}\frac{1}{\delta} - \frac{\log_{\frac{1-\gamma}{\epsilon}} 2}{2}\log_{\frac{1-\epsilon}{\gamma}}\frac{1}{\delta} - \frac{1}{4}\log_{\frac{1-\gamma}{\epsilon}}\frac{1}{\delta} .$$

As elaborated in the full paper, the lower bound part of Theorem 4.1 for the case $\delta_1 = \delta_2 = \delta$, follows from Corollary 4.7.

4.3 The Upper Bound: An Algorithm

Our description of the algorithm proceeds in two steps.

Algorithm for $\gamma = \epsilon$. At any time in the course of the algorithm, each strand has a *count*, defined to be equal to $i - j$ if the strand was involved in $i + j$ Extracts, i of which classified the strand as a 1 and j of which classified the strand as a 0. The algorithm proceeds in phases. Let $D = \lceil \log_{1/\epsilon} 1/\delta \rceil$.

Initially: Only one tube, of count 0.
Repeat the following $4D$ times:

- Perform an Extract on each test tube present at the beginning of the phase.
- Merge the tubes whose strands have the same count.

Output: all tubes with positive count are classified as 1, and all the others as 0.

CORRECTNESS OF THE ALGORITHM The analysis is straightforward: Each strand behaves like a random walk biased either $(\epsilon, 1-\epsilon)$ or $(1-\epsilon, \epsilon)$. By symmetry, assume the strand is a 1 and so is biased upwards. After $4D$ phases, it has been involved in $4D$ Extracts, and the probability that its count is non positive is $O(\epsilon^D) = O(\delta)$ (follows immediately from Chernoff inequality).

EFFICIENCY OF THE ALGORITHM In the beginning of the ith phase, i tubes are present. In each phase of the $4D$ phases, the algorithm performs a single Extract on each tube present. Thus, altogether the algorithm performs $4D(4D+1)/2 = 4\lceil \log_\epsilon \delta \rceil (4\lceil \log_\epsilon(\delta) \rceil + 1)/2$ Extracts.

Refer to the above algorithm as *Algorithm A*.

To modify the algorithm for the case that $\gamma \neq \epsilon$, replace each Extract of the above algorithm by a *Super Extract* in which the probabilities for misclassifying a zero and a one are approximately equal. The fact that using several Extracts one can make mis-classifications in both directions approximately equal was observed also in [1].

By symmetry, assume $\gamma \leq \epsilon$.

Super Extracts To perform a Super Extract on test tube T, one proceeds in $\lceil \log_\epsilon \gamma \rceil$ steps.

Initially there is one tube called T.

Repeat the following $\lceil \log_\epsilon \gamma \rceil$ times:

- Perform an Extract on each tube present at the beginning of the phase.
- Merge all tubes that contain strands that were classified as one in at least one of the phases. (As a result we are left with two tubes.)

Output: The tube containing strands that were classified as one in at least one of the phases is classified as one; the other tube is classified as zero.

At the end of the Super Extract on tube T we have two tubes, $+ST, -ST$ so that $+ST$ contains all strands that were classified as one by at least one of the $\lceil \log_\epsilon \gamma \rceil$ Extracts performed on them, and $-ST$ contains all strands that were classified as zero by each of the $\lceil \log_\epsilon \gamma \rceil$ Extracts performed on them.

Let ϵ', γ' be the probabilities that a one and a zero bit will be misclassified by the Super Extract, respectively. Clearly, $\gamma' \leq \gamma\lceil \log_\epsilon \gamma \rceil$, and $\epsilon' \leq \gamma$.

Final Algorithm Replace each Extract in algorithm A by a Super Extract, and redefine $D = \lceil \log_{\gamma'} \delta \rceil$.
CORRECTNESS Note that $\gamma' \geq \epsilon'$. Thus the correctness of the final algorithm follows the same reasoning as in Algorithm A.
EFFICIENCY As argued in Algorithm A, there are $4D(4D+1)/2$ Super Extracts. Each Super Extract consists of exactly $2\lceil \log_\epsilon \gamma \rceil - 1$ Extracts. Thus, the total number of Extracts is:

$$4D(4D+1) \cdot (2\lceil \log_\epsilon \gamma \rceil - 1) = 4\lceil \log_{\gamma'} \delta \rceil (4\lceil \log_{\gamma'} \delta \rceil + 1) \cdot (2\lceil \log_\epsilon \gamma \rceil - 1) \ .$$

After some algebraic manipulations, and also using the fact that $\gamma + \epsilon < 1$, we get the upper bound part of Theorem 4.1 for $\delta_1 = \delta_2$.

5 Error-Resilient Function Evaluation

We are given an initial tube containing a number of strands, each of which encodes a sequence of bits $x = (x_1, x_2, \cdots, x_n)$. Let $f(x_1, x_2, \ldots, x_n)$ be a boolean function of n variables.

The function evaluation problem comes in two variants.

5.1 The Weak Variant

In this variant the goal is to divide the strands into two output tubes, T_0 and T_1, such that:

1. each strand x for which $f(x) = 1$ has probability at most δ of ending up in T_0; and
2. each strand x for which $f(x) = 0$ has probability of at most δ of ending up in T_1.

5.2 The Strong Variant

The second variant of the problem is called the *strong* variant. Let $d(x)$ be equal to the minimum number of bits of x which must be flipped to change the value of $f(x)$. The goal is to divide the strands into two output tubes, T_0 and T_1, such that

1. each strand x for which $f(x) = 1$ has probability at most $\delta^{d(x)}$ of ending up in T_0; and
2. each strand x for which $f(x) = 0$ has probability of at most $\delta^{d(x)}$ of ending up in T_1.

The motivation for the strong variant is as follows. Often the goal in experiments is to simply output two tubes T_0 and T_1 such that a random strand from T_1 satisfies f with probability at least $1 - \delta'$. If the initial distribution is uniform, *i.e.* each bit is 1 with probability 1/2, then this experimental goal is obtained by using the model above with $\delta = \delta'/(n(1 + \delta'))$.

6 Bounds for Error-Resilient Function Evaluation

In this section we show how our analysis for error-resilient bit evaluation can be extended for error-resilient function evaluation. Our lower bounds apply even for the weak variant of the problem, and our algorithms apply even for for the strong variant.

6.1 The Upper Bound: Error-Resilient Function Evaluation

This section provides a simple transformation from the idealized error-free model, in which the Extract operation is perfect, to the more realistic noisy model, in which Extracts may provide wrong results. The transformation is efficient. In other words, we show how algorithms can be made error-resilient without a big price in their efficiency.

Formally, refer to an idealized setting in which Extract always produces the correct result as the *error-free model*. That is, in the error-free model, performing an Extract on the ith bit of strands in test tube T results in two test tubes, $+T, -T$, such that $+T$ consists of all strands x in which x_i is 1, and $-T$ consists of all other strands. An *algorithm for f in the error-free model* is an algorithm that, in the absence of errors in the Extract operation, outputs two test tubes T_0, T_1, such that T_0 contains all strands x for which $f(x) = 0$, and T_1 contains all strands x for which $f(x) = 1$.

Let A be an algorithm for f in the error-free model. Call an Extract from tube T on bit x_i *redundant* if it is implied by the results of previous Extracts either that every strand in T has $x_i = 0$ or that every strand in T has $x_i = 1$. Assume that A performs no redundant Extracts. (otherwise it can be modified by eliminating the redundant Extracts.) If A performs $a(n)$ Extracts, then we can construct an algorithm for strong error-resilient evaluation of function f that performs $O(a(n) \cdot \lceil \log_{1/\epsilon} \frac{a(n)}{\delta} \rceil \lceil \log_{1/\gamma} \frac{a(n)}{\delta} \rceil)$ Extracts, as follows:

Error-Resilient Algorithm for f: *Whenever A does "Extract x_i", do instead our error-resilient bit evaluation algorithm for computing x_i with error probability $\delta/a(n)$ (See Section 4.3.)*

Refer to the resulting algorithm as algorithm B.

Theorem 6.1 *There is a transformation that transforms any algorithm for function evaluation into an error-resilient algorithm such that if the original algorithm performs $a(n)$ Extracts, then the error-resilient algorithm performs $O(a(n)\lceil \log_{\frac{1}{\epsilon}} \frac{a(n)}{\delta} \rceil \lceil \log_{\frac{1}{\gamma}} \frac{a(n)}{\delta} \rceil)$ Extracts.*

The proof consists of showing that Algorithm B is the required error-resilient algorithm.

Observe that Conjunction, Disjunction and Parity all have $O(n)$ algorithms in the error-free model. Thus,

Corollary 6.2 *The above transformation yields error-resilient algorithms for computing Conjunction, Disjunction and Parity, that perform $O(n \cdot \lceil \log_{\frac{1}{\gamma}} \frac{n}{\delta} \rceil \cdot \lceil \log_{\frac{1}{\epsilon}} \frac{n}{\delta} \rceil)$ Extracts.*

6.2 Lower Bounds for Error-Resilient Function Evaluation

Corollary 6.2 showed error-resilient algorithms for computing Conjunction, Disjunction and Parity, that perform $O(n \cdot \lceil \log_{\frac{1}{\gamma}} \frac{n}{\delta} \rceil \cdot \lceil \log_{\frac{1}{\epsilon}} \frac{n}{\delta} \rceil)$ Extracts. In this section we show a lower bound.

Define the *all zeroes strand problem* as follows: Given a set of strands, output two tubes T_0, T_1, such that each strand that consists solely of zeroes will end up in T_1 with probability $\geq 1 - \delta$, and each strand that is not all zeroes, will end up in T_0 with probability $\geq 1 - \delta$.

The proof of the following lemma is given in the Appendix.

Lemma 6.3 *The number of Extract operations required by a weak error-resilient algorithm for the all zeroes strand problem with error probability* δ *is* $\Omega(n \cdot \lceil \log_{\frac{1-\epsilon}{\gamma}} \frac{1}{\delta} \rceil \cdot \lceil \log_{\frac{1-\gamma}{\epsilon}} \frac{1}{\delta} \rceil)$.

It is immediate that the lower bound also applies to strong error-resilient algorithms for the same problems.

Similarly we get:

Corollary 6.4 *The number of Extract operations required by weak error-resilient algorithms for Conjunction, Disjunction and Parity with error probability* δ *is* $\Omega(n \cdot \lceil \log_{\frac{1-\epsilon}{\gamma}} \frac{1}{\delta} \rceil \cdot \lceil \log_{\frac{1-\gamma}{\epsilon}} \frac{1}{\delta} \rceil)$.

Observe that for $\delta = O(1/n)$ the bounds in Corollaries 6.2 and 6.4 are matching. In practice, δ will usually be at most $1/n$ since otherwise the number of bad strands in the resulting tube T_1, *i.e.* the tube that is supposed to contain mostly strands that satisfy f, will overwhelm the number of good strands there, since the initial number of bad strands is much higher than the number of good strands.

References

[1] L. Adleman. Molecular computation of solutions to combinatorial problems. In *Science* 266:1021, Nov. 11, 1994.

[2] D. Beaver. Nanocomputing: The DNA solution. To appear in *Journal of computational Biology.* Preliminary abstract appeared as 'Factoring: The DNA solution', in *Advances in Cryptography - Asiacrypt '94 Proceedings,* Springer Verlag LNCS, 1994.

[3] D. Beaver. A universal molecular computer. Technical Report CSE-95-001, Penn State University, 1995.

[4] D. Beaver. Computing with DNA. Manuscript 1995.

[5] D. Boneh, C. Dunworth, and R. J. Lipton. Breaking DES using a molecular computer. Technical Report CS-TR-489-95, Princeton University, 1995.

[6] D. Boneh, C. Dunworth, R. J. Lipton, and J. Sgall. On the computational power of DNA. Manuscript 1995.

[7] D. Boneh and R. J. Lipton. Making DNA computers error resistant. Technical Report CS-TR-491-95, Princeton University, 1995.

[8] U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Computing with unreliable information. In *Proceedings of the 22nd ACM Symposium on Theory of Computing*, pages 128–137, 1990.

[9] A. Gal. Lower bounds for the complexity for reliable boolean circuits with noisy gates. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 594–601, October 1991.

[10] C. Kenyon and V. King. On boolean decision trees with faulty nodes. In *Random Structures and Algorithms*, page 453–464, Vol. 5, No. 3, 1994.

[11] R. M. Karp, C. Kenyon, and O. Waarts. Error-resilient DNA computation. Manuscript 1995.

[12] R. J. Lipton. Using DNA to solve NP-Complete problems. In *Science*: 268:542-545, April 28, 1995.

[13] National Bureau of Standards. Data Encryption Standard. U.S. Department of Commerce, FIPS, pub. 46, January 1977.

[14] N. Nisan. CREW PRAMs and decision trees. In *Proceedings of the 21st ACM Symposium on Theory of Computing*, pages 327–335, 1989.

[15] N. Pippenger. On networks of noisy gates. In *Proceedings of the 26th IEEE Symposium on Foundations of Computer Science*, pages 30-38, 1985.

[16] R. Reischuk and B. Schmeltz. Reliable computation with noisy circuits and decisions trees–A general $n \log n$ lower bound. In *Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science*, pages 602–611, 1991.

[17] R. L. Rivest, A. R. Meyer, D. J. Kleitman, K. Winklmann, and J. Spencer. Coping with errors in binary search procedures. In *Journal of Computing and Systems Sciences*, Vol. 20, pages 396–404, 1980.

[18] D. Rooß and K. W. Wagner. On the power of Bio-Computers. Manuscript 1995.

A Appendix

Recall that by a *fraction α of good strands in a tube*, we mean that the mass of good strands in this tube consitutes an α fraction of the total mass of good strands.

Proof of Lemma 4.3

Proof: Assume we do an Extract on a test tube T which contains expected fraction α of good strands and its likelihood is c. Define $logc = \log_{\frac{\gamma}{1-\epsilon}} c$.

We will first compute the change in return(l). Observe that the Extract changes the likelihoods only of strands in T, and hence the only change in μ is the one caused by the change of likelihoods of the strands in T. Denote return(l) just before and just after the Extract by return(l)$before$ and return(l)$after$ respectively.

Case I: $l \geq logc + 1$. The contribution of the strands in T to return(l)$before$ is contained in the term $\int_{-\infty}^{l} d\mu(x)$ of the expression that computes return(l)$before$. Thus this contribution is α. Claim 4.2 implies that each of $+T$ and $-T$ has likelihood at least $\frac{\gamma}{1-\epsilon} \cdot c$. (Note that $\frac{\gamma}{1-\epsilon} < \frac{1-\gamma}{\epsilon}$ due to the assumption that $\epsilon + \gamma < 1$.) Hence each of $+T$ and $-T$ has

log-likelihood at most l. Thus, the sum of the contributions of the strands in $+T$ and $-T$ to return$(l)after$ is contained in $\int_{-\infty}^{l} d\mu(x)$, and is again α. Hence return$(l)after$ − return$(l)before = \alpha - \alpha = 0$.

Case II: $l < logc - \log_{\frac{\gamma}{1-\epsilon}} \frac{\epsilon}{1-\gamma}$**.** Note that $\log_{\frac{\gamma}{1-\epsilon}} \frac{\epsilon}{1-\gamma}$ is nonnegative since $\frac{\gamma}{1-\epsilon}$ and $\frac{\epsilon}{1-\gamma}$ are both smaller than 1 because of the assumption that $\epsilon + \gamma < 1$.

The contribution of the strands in T to return$(l)before$ is contained in the term $\int_{l+}^{\infty} (\gamma/(1-\epsilon))^{x-l} d\mu(x)$ of the expression that computes return$(l)before$. Thus, this contribution is $\alpha \cdot (\frac{\gamma}{1-\epsilon})^{logc-l}$. By Claim 4.2, $+T$ contains a fraction $(1-\epsilon)\alpha$ of good strands and its likelihood is $\frac{\gamma}{1-\epsilon} \cdot c$. Observe that the contribution of the strands in $+T$ to return$(l)after$ is contained in the term $\int_{l+}^{\infty} (\gamma/(1-\epsilon))^{x-l} d\mu(x)$. Thus the contribution of $+T$ to return$(l)after$ is $\alpha(1-\epsilon) \cdot (\frac{\gamma}{1-\epsilon})^{logc+1-l}$. By Claim 4.2, $-T$ contains a mass $\epsilon\alpha$ of good strands and its likelihood is $\frac{1-\gamma}{\epsilon} \cdot c$. Observe that also the contribution of the strands in $-T$ to return$(l)after$ is contained in the term $\int_{l+}^{\infty} (\gamma/(1-\epsilon))^{x-l} d\mu(x)$. Thus the contribution of $-T$ to return$(l)after$ is $\alpha\epsilon \cdot (\frac{\gamma}{1-\epsilon})^{logc - \log_{\frac{\gamma}{1-\epsilon}} (\frac{\epsilon}{1-\gamma}) - l} = \alpha\epsilon \cdot (\frac{\gamma}{1-\epsilon})^{logc-l} \cdot \frac{1-\gamma}{\epsilon}$. Hence,

$$\begin{aligned} &\text{return}(l)after - \text{return}(l)before = \\ &= \alpha \left(\frac{\gamma}{1-\epsilon}\right)^{logc-l} \cdot \left((1-\epsilon)\frac{\gamma}{1-\epsilon} + \epsilon\frac{1-\gamma}{\epsilon} - 1\right) \\ &= 0 \ . \end{aligned}$$

Case III: $logc - \log_{\frac{\gamma}{1-\epsilon}} \frac{\epsilon}{1-\gamma} \leq l < logc$**.** Denote by g the contribution to return$(l)before$ of all good strands other than those in T. Note that g is also the sum of the contributions to return$(l)after$ of all strands that are neither in $+T$ nor in $-T$.

The contribution of the strands in T to return$(l)before$ is: $\alpha \cdot (\frac{\gamma}{1-\epsilon})^{logc-l}$. The contribution of the strands in $+T$ to return$(l)after$ is $(1-\epsilon)\alpha \cdot (\frac{\gamma}{1-\epsilon})^{logc+1-l}$. The contribution of the strands in $-T$ to return$(l)after$ is: $\epsilon\alpha$. Thus,

$$\frac{\text{return}(l)after}{\text{return}(l)before} = \frac{g + \alpha(1-\epsilon)(\frac{\gamma}{1-\epsilon})^{logc+1-l} + \alpha\epsilon}{g + \alpha(\frac{\gamma}{1-\epsilon})^{logc-l}}$$

$$\begin{aligned} &\geq \frac{\alpha(1-\epsilon)(\frac{\gamma}{1-\epsilon})^{logc+1-l}+\alpha\epsilon}{\alpha(\frac{\gamma}{1-\epsilon})^{logc-l}} \\ &= \gamma\frac{(\frac{\gamma}{1-\epsilon})^{logc-l}+\frac{\epsilon}{\gamma}}{(\frac{\gamma}{1-\epsilon})^{logc-l}} \\ &\geq \gamma+\epsilon\ . \end{aligned}$$

The last inequality follows since $\gamma/(1-\epsilon) < 1$ (because $\epsilon+\gamma<1$), and by this case assumption $logc - l > 0$.

Case IV: $logc \leq l < logc+1$**.** Denote by g the contribution to return$(l)before$ of all strands other than those in T. Clearly, the sum of the contributions to return$(l)after$ of the strands that are neither in $+T$ nor in $-T$ is also g.

The contribution of the strands in T to return$(l)before$ is α. The contribution of the strands in $+T$ to return$(l)after$ is $(1-\epsilon)\alpha\cdot(\frac{\gamma}{1-\epsilon})^{logc+1-l}$. The contribution of the strands in $-T$ to return$(l)after$ is: $\epsilon\alpha$. Thus,

$$\begin{aligned} \frac{\text{return}(l)after}{\text{return}(l)before} &= \frac{g+\alpha(1-\epsilon)(\frac{\gamma}{1-\epsilon})^{logc+1-l}+\alpha\epsilon}{g+\alpha} \\ &\geq \frac{\alpha(1-\epsilon)(\frac{\gamma}{1-\epsilon})^{logc+1-l}+\alpha\epsilon}{\alpha} \\ &= \gamma\left(\left(\frac{\gamma}{1-\epsilon}\right)^{logc-l}+\frac{\epsilon}{\gamma}\right) \\ &\geq \gamma+\epsilon\ . \end{aligned}$$

The last inequality follows since $\gamma/(1-\epsilon)<1$ and by this case assumption $logc - l < 0$.

Thus, the change in Ψ is contributed by the return(l)s in Cases III and IV. Denote by $\Psi before$ and $\Psi after$ the value of Ψ just before and just after the Extract respectively. Then,

$$\begin{aligned} &\Psi after - \Psi before \geq \\ &\geq \left(1+\log_{\frac{1-\epsilon}{\gamma}}\frac{1-\gamma}{\epsilon}\right)\cdot\log_{\frac{1-\gamma}{\epsilon}}(\gamma+\epsilon) \\ &= \log_{\frac{1-\gamma}{\epsilon}}(\gamma+\epsilon)+\log_{\frac{1-\epsilon}{\gamma}}(\gamma+\epsilon) \\ &\geq -2\ . \end{aligned}$$

The last inequality follows because, since $\gamma+\epsilon<1$, we have: $\gamma+\epsilon \geq \frac{\gamma}{1-\epsilon}, \frac{\epsilon}{1-\epsilon}$.

■

Proof of Lemma 4.4

Proof: Without loss of generality assume that a Merge only acts on two tubes of equal mass of good strands (otherwise we can do a sequence of Merge operations). Thus, the Merge takes a mass that contains a fraction α of good strands at log-likelihood x and a mass that contains a fraction α of good strands at log-likelihood $y \geq x$ and Merges them into a mass containing a fraction 2α of good strands at log-likelihood z, where

$$\left(\frac{\gamma}{1-\epsilon}\right)^z = \frac{1}{2}\left(\frac{\gamma}{1-\epsilon}\right)^x + \frac{1}{2}\left(\frac{\gamma}{1-\epsilon}\right)^y .$$

Let us study how return(l) changes. Denote by return$(l)before$ and return$(l)after$ the value of return(l) just before and just after the Merge respectively.

If $l \geq y$, return$(l)after$ = return$(l)before$.

If $l < x$, then

$$\text{return}(l)after - \text{return}(l)before = 2\alpha(\frac{\gamma}{1-\epsilon})^{z-l} - \alpha(\frac{\gamma}{1-\epsilon})^{x-l} - \alpha(\frac{\gamma}{1-\epsilon})^{y-l} = 0.$$

If $x \leq l < z$, then

$$\frac{\text{return}(l)after}{\text{return}(l)before} = \frac{2\alpha(\frac{\gamma}{1-\epsilon})^{z-l}}{\alpha + \alpha(\frac{\gamma}{1-\epsilon})^{y-l}} = \frac{(\frac{\gamma}{1-\epsilon})^{x-l} + (\frac{\gamma}{1-\epsilon})^{y-l}}{1 + (\frac{\gamma}{1-\epsilon})^{y-l}} \geq 1$$

since $x - l \leq 0$ and $\gamma/(1-\epsilon) < 1$.

If $z \leq l < y$, then

$$\frac{\text{return}(l)after}{\text{return}(l)before} = \frac{2\alpha}{\alpha + \alpha(\frac{\gamma}{1-\epsilon})^{y-l}} = \frac{2}{1 + (\frac{\gamma}{1-\epsilon})^{y-l}} \geq 1$$

since $y - l \geq 0$ and $\gamma/(1-\epsilon) < 1$.

So for all l, a Merge does not decrease return(l), and so Ψ does not decrease. ■

Proof of Lemma 6.3

Proof: Consider the following weaker problem. We are given n copies of $n+1$ types of strands, such that for $1 \leq i \leq n$, a strand of the ith type has a

1 only in the ith bit, and a strand of the $n+1$st type is the all zeroes strand. The strands are given in n tubes: An oracle tells us that for $1 \leq i \leq n$, the ith tube, G_i, contains all strands of type i and one strand of type $n+1$ *i.e.* the all zeroes strand. The goal is again to output two test tube, T_0, T_1 such that the all zeroes strand end up in T_1 with probability at least $1-\delta$, and all other strands end up in T_0 with probability at least $1-\delta$.

Clearly the above problem is not harder than the problem of detecting the all zeroes strand, since we can start the algorithm by merging all tests tubes, with no additional cost. Thus, it is enough to prove the lower bound for the weaker problem.

Focus on the weaker problem. First we claim:

Claim A.1 *Merging strands that originate in different tubes G_i, G_j will not reduce the number of necessary Extracts.*

Proof: Consider the first Merge of strands that originated from different initial tubes, say tubes $G_{i_1}, \ldots, G_{i_k}$. Call the resulting tube G. Let E be the first Extract done on tube G after the Merge. If this Extract is done on bit $j \notin \{i_1, \ldots, i_k\}$, then cancel this Extract since we know what its result should be (*i.e.* zero). Otherwise, we can postpone the Merge, and perform the Extract only on the strands that originated from G_j and that participated in the Merge. ■

Claim A.1 implies that we should perform separate Extracts on each of the original tubes, and separate the strands given in the ith test tubes into two tubes, $+G_i, -G_i$, such that $+G_i$ contains all strands of G_i where the ith bit is 1, and $-G_i$ contains all other strands of G_i. Each strand in G_i should end up in the correct $+G_i, -G_i$ with probability at least $1-\delta$. The desired tubes T_1 will be obtained by simply merging in the end the $-G_i, i = 1, \ldots, n$ and T_0 will be obtained by merging the $+G_i, i = 1, \ldots, n$.

However, Theorem 4.1 implies that, for each G_i, the number of Extracts required for separating it to $+G_i, -G_i$ is $\Omega(\lceil \log_{\frac{1-\epsilon}{\gamma}} \frac{1}{\delta} \rceil \cdot \lceil \log_{\frac{1-\gamma}{\epsilon}} \frac{1}{\delta} \rceil)$. The claim follows from the fact that we have n initial tubes, $G_1, \ldots, G_n$. ■