Breaking DES Using a Molecular Computer

(Extended Abstract)

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

Abstract

Recently Adleman has shown that a small traveling salesman problem can be solved by molecular operations. In this paper we show how the same principles can be applied to breaking the Data Encryption Standard (DES). We describe in detail a library of operations which are useful when working with a molecular computer. We estimate that given one arbitrary (plain-text, cipher-text) pair, one can recover the DES key in about 4 months of work. Furthermore, we show that under chosen plain-text attack it is possible to recover the DES key in one day using some preprocessing. Our method can be generalized to break any cryptosystem which uses keys of length less than 64 bits.

Keywords: DNA-computing, Molecular-computing, DES.

*Supported in part by NSF CCR–9304718.

1 Introduction

Due to advances in molecular biology it is nowadays possible to create a soup of roughly 10^{17} DNA strands that fits in a liter of water. Adleman [1] has shown that each DNA strand can be used to perform computations. Thus, a small test tube containing DNA strands seems to have more computing power than the most powerful parallel computers. The drawback of this approach is that basic operations using DNA take a long time (e.g. one hour). Thus, we are capable of performing 10^{17} basic operations at once, though each operation requires several hours to complete. Throughout the paper we will refer to operations done on DNA strands as bio-steps. This will be explained in more detail in Section 2.

Recently, Lipton [9] has come up with an encoding scheme that enabled him to solve the satisfiability problem of formulas with a small number of variables. A generalization of this scheme [3] can be used to solve the satisfiability problem for circuits. Using these methods, the number of bio-steps required to find a satisfying assignment is proportional to the size of the circuit. Lipton's approach raises the hope of using molecular computing to solve hard problems that come up in practice. However, to make this procedure practical one needs to encode the problem at hand so as to minimize the number of bio-steps required to solve it.

In this paper we give the first example of a "real world" hard problem that can be solved using a molecular computer. We present a "molecular program" for breaking the Data Encryption Standard [5], or DES for short. DES is a widely used encryption procedure. It encrypts 64 bit messages and uses a 56 bit key. By breaking DES we mean that given one (plain-text, cipher-text) pair we can find a key mapping the plain-text to the cipher-text. Furthermore, we show that using chosen plain-text attack[1] it is possible to recover the key in one day of work using some preprocessing. This means that after the preprocessing work is done it is possible to break many DES systems for very little work.

Several researchers [11] have come up with various methods for implementing non-deterministic Turing Machines using molecular computers. Clearly a non-deterministic Turing Machine can break any crypto-system, including DES, by guessing the correct key. Though these results are very important theoretically, they are not useful in practice. Breaking DES by simulating a non-deterministic Turing Machine would require millions of biological operations. Running such experiments would take hundreds of years. We note that recently Adleman et al. [2] developed an alternate approach for breaking DES using DNA computations.

It should be pointed out that attacks using differential cryptanalysis methods have proven to be very useful for breaking DES [6, 10]. The central achievement of these attacks is that DES can be broken on a conventional computer in 2^{43} steps. This means that DES can be broken within several days on a conventional computer. Note however, that the differential cryptanalysis methods require 2^{43} pairs of (plain-text, cipher-text), while our molecular computer requires only one such pair.

Another conventional attack on DES was suggested by Wiener [12]. Wiener predicts that using dedicated hardware which costs \$1 million it is possible to search through all 2^{56} DES keys in 7 hours. Wiener's attack makes use of the special structure of the DES circuit. On the other hand, the attack using a molecular computer is far more general. It can be used to break any cryptosystem which uses keys of length less than 64 bits provided the encryption circuit is not too big. Furthermore, if DES was made to use 64 bit keys, Wiener's attack would slow down by a factor of 256, while the molecular attack would be unaffected.

[1] Chosen plain-text attack means that an adversary can obtain a (plain-text, cipher-text) pair where he/she can specify the plain-text.

Throughout this paper we will assume an error free model. This means that all the molecular biology experiments work perfectly without any errors. For now, this assumption is valid since we view the problem of coping with errors as tangential to that of finding applications for molecular machines. Furthermore, we are confident that errors that arise during the operation of a molecular computer can be dealt with. This is based on our experience with conventional electronic machines where fault-tolerance techniques enable one to perform error-free computations in the presence of faulty processors. We note that several researchers [4, 8] have already suggested some techniques for making molecular computers resistant to errors.

2 Overview of molecular computing

In order to understand the computation described here, it is necessary to gain a fundamental understanding of DNA's structure and function. DNA (deoxyribonucleic acid) is found in every living creature as the storage medium for genetic information. It is comprised of subunits called nucleotides that are strung together into polymer chains (the chemical structure of a single nucleotide is shown in Figure 1). DNA polymer chains are more commonly called DNA *strands*, and short strands are called *oligonucleotides*, or simply *oligos*.

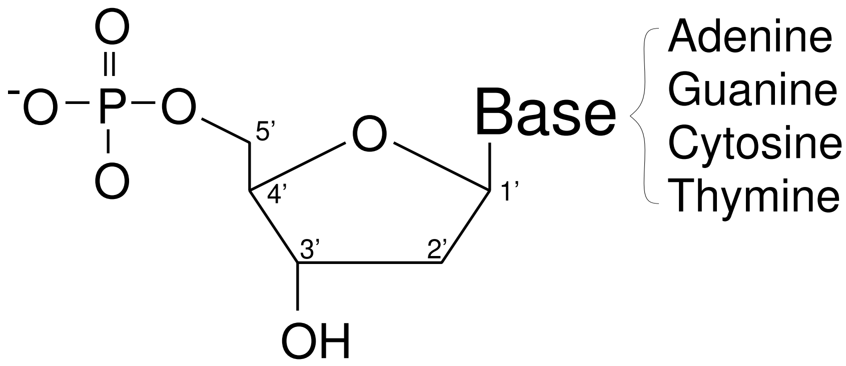

Figure 1: Chemical structure of a nucleotide

There are four kinds of nucleotides in DNA, distinguished by the chemical group, or base, attached to it. The four bases are adenine, guanine, cytosine, and thymine, abbreviated as `A`, `G`, `C`, and `T` (we will henceforth use these letters to refer to nucleotides containing these bases). Single nucleotides are linked together end-to-end to form DNA strands in a process called polymerization. This linking occurs via a reaction between the 5' phosphate of one nucleotide and the 3' hydroxyl of another. Note that every DNA strand will have two distinct ends—one with a free 5' PO_4 group and the other with a free 3' OH group, referred to as the 5' and 3' ends, respectively. Thus, we can think of the sequence of nucleotides in any strand as having a natural orientation. This orientation is important to remember because it restricts the kinds of operations that one can perform on the strands.

DNA does not usually exist in nature as free single strands, though. Under appropriate conditions single strands will pair up and twist around each other, forming the famous double helix structure discovered by Watson and Crick. This pairing occurs because of a mutual attraction, called hydrogen bonding, that exists between `A`s and `T`s and between `G`s and `C`s. Hence, `A/T` and `G/C` are called *complementary* base pairs. A further condition is that hydrogen bonding will only occur if the pair of complementary strands are oriented in an *antiparallel* fashion. That is, as one strand of a double helix runs in the 5' → 3' direction, its complement runs in the 3' → 5' direction. For example, if we have the strands `5'-AAGCGTAG-3'`, `3'-TTCGCATC-5'`, and `5'-TTCGCATC-3'`, the first two can pair up into a double strand—namely, $\genfrac{}{}{0pt}{}{\texttt{5'-AAGCGTAG-3'}}{\texttt{3'-TTCGCATC-5'}}$ —but the first and last ones cannot, because when lined up in an antiparallel fashion we see that they are not complementary: $\genfrac{}{}{0pt}{}{\texttt{5'-AAGCGTAG-3'}}{\texttt{3'-CTACGCTT-5'}}$. This process of complementary strands coming together to form a double helix is called *annealing*, and the reverse process—a double helix coming apart to yield its two constituent single strands— is called *melting*.

2.1 Notation

We have devised a notation for abstractly representing DNA strands in order to succinctly describe the changes they undergo during various operations. First, we represent strings over the alphabet {`A`,`C`,`G`,`T`} using letters for variable names (e.g. x = `ACCTGAC`). Note that x is a *string*, but not yet a *strand*—that is, it's just a sequence of characters, with no orientation implied.

The *complement* of a string x, denoted by $\hat{x}$, is the string that results when each character of x has been replaced by its Watson-Crick complement. That is, we apply the following mapping to every character in x: `A`→`T`, `C`→`G`, `G`→`C`, `T`→`A`. Thus, for x = `ACCTGAC`, we have $\hat{x}$ = `TGGACTG`. The *reverse* of a string x, denoted x^R, is just the string x in reverse order. Thus for the same example, x^R = `CAGTCCA`.

For a string x we denote by $\upharpoonright x$ the single DNA strand whose sequence is x and whose orientation is $5' \rightarrow 3'$, as x is read from left to right. For example, if x = `ACCTGAC`, then $\upharpoonright x$ is the strand `5'-ACCTGAC-3'`. Similarly, $\downharpoonleft x$ denotes the strand complementary to $\upharpoonright x$. For our example, $\downharpoonleft x$ = `3'-TGGACTG-5'`. Finally, we have the operator "$\upharpoonright\!\!\downharpoonleft$". $\upharpoonright\!\!\downharpoonleft x$ denotes the double strand that results when $\upharpoonright x$ and $\downharpoonleft x$ anneal in solution.

The table below summarizes these operators:

$$
\begin{array}{rcll}
x & = & \texttt{ACCTGAC} & \\
\hat{x} & = & \texttt{TGGACTG} & \\
x^R & = & \texttt{CAGTCCA} & \\
\upharpoonright x & = & \texttt{5'-ACCTGAC-3'} & \\
\downharpoonleft x & = & \texttt{3'-TGGACTG-5'} & \qquad (\text{note: } \upharpoonright \hat{x}^R = \texttt{5'-GTCAGGT-3'} = \downharpoonleft x) \\
\upharpoonright\!\!\downharpoonleft x & = & \genfrac{}{}{0pt}{}{\texttt{5'-ACCTGAC-3'}}{\texttt{3'-TGGACTG-5'}} &
\end{array}
$$

The convenience of this notation becomes clearer when we try to describe various operations on DNA strands in solution. For example, the formula

$$\upharpoonright x \; + \; \downharpoonleft x \quad \longrightarrow \quad \upharpoonright\!\!\downharpoonleft x$$

describes the process of annealing. As another example, we see that the strand $\upharpoonright xy$ and the strand $\downharpoonleft yz$ will anneal under appropriate conditions to form the expected partial duplex $\upharpoonright x \upharpoonright\!\!\downharpoonleft y \downharpoonleft z$.

2.2 Biological operations

Our fundamental model of computation will be to apply a sequence of operations to a set of strands in a test tube. The operations that we use are derived from a collection of experiments commonly used in molecular biology today [1].

2.2.1 Extract

We need the ability to extract from a test tube all strands that contain some specific short nucleotide sequence x. To accomplish this, we use the biotin-streptavidin bead separation technique described in [1]. In short, one can create a solution containing many copies of the oligo $\downarrow x$, each bound to the surface of tiny magnetic beads. To this bead-oligo mixture we add our DNA strands. Only those strands containing the sequence x will anneal to the $\downarrow x$ oligos, and these annealed strands can then be separated from the rest magnetically. We refer to this operation as *extract* using beads of type $\downarrow x$.

2.2.2 Amplification via PCR

At times we must make duplicates of all the DNA strands in a test tube, which can be accomplished via a process called the polymerase chain reaction (PCR). For PCR to work, every strand must be of the form $\updownarrow aBc$, where a and c are fixed primers and B is arbitrary. In addition, neither a nor c can occur as substrings of B. Since our encoding of computational strands satisfies these requirements (see Section 3), PCR should successfully replicate every strand in the tube. We will call this operation *amplify*.

2.2.3 Tag

Sometimes we will need to append a new short sequence z onto the end of every strand in a tube. If every strand is of the form $\uparrow Xy$, where X is arbitrary and y is fixed, then this append can be accomplished in two steps, generating the desired strand $\uparrow Xyz$. First, we anneal the oligo $\downarrow yz$ to each strand, creating the partial duplex $\uparrow X \updownarrow y \downarrow z$. Next, we add the enzyme DNA polymerase, which will convert partial duplexes into full duplexes by "reading" the single-stranded portions and building their Watson-Crick complements, one nucleotide at a time. Thus, we can convert our partial duplexes $\uparrow X \updownarrow y \downarrow z$ into the desired full duplex $\updownarrow Xyz$. We call this operation *tagging*.

3 Representing binary strings

In this section we describe how to encode binary strings as DNA molecules. This encoding enables us to use the biological operations of Section 2 to perform logical operations on binary strings.

Let $x = x_1 \ldots x_n$ be an n-bit binary string. The idea is to assign a unique 30-mer (a 30 long DNA sequence) to each bit position and bit value. One can regard these 30-mers as strings of length 30 over the alphabet $\{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$.

1. Let $B_i(0)$ be the oligo used to encode the fact that the i-th bit of x is 0. Similarly, let $B_i(1)$ be the oligo used to encode the fact that the i-th bit is 1.

2. For $i = 0, \ldots, n$ let S_i be a 30-mer used as a separator between consecutive bits.

The DNA strand representing the binary string $x \in \{0,1\}^n$ is:

$$\updownarrow S_0\ B_1(x_1)\ S_1\ B_2(x_2)\ S_2\ , \ldots, S_{n-1}\ B_n(x_n)\ S_n$$

It is crucial that the strings $B_i(x)$ and S_i are all distinct. Furthermore, it is desirable that any pair of these strings not contain long common substrings. For example, there shouldn't be a string of length 15 which is a substring of both S_0 and S_1. This can be achieved by choosing the words $B_i(b)$ and S_i at random, since the length of the longest common substring of two random strings tends to be short.

Our choice of using strands of length 30 to represent the bits is driven by two factors. First, the longer the strands are the more expensive the experiment becomes. Thus we would like the strands to be as short as possible. Second, the strands should not share long common substrings. Adleman in his experiment used 20-mers to encode the data. Since our DES experiment is two orders of magnitude bigger than Adleman's we estimate that using 30-mers is sufficient. Observe that using 30-mers gives us a million times more strands to choose from since $4^{30} \approx 10^6 4^{20}$.

The encoding we propose is useful for doing PCR. Let T be solution of DNA strands representing a set of n-bit binary strings. By definition, all strands contain S_0 on one end and S_n on the other. Consequently, S_0 and S_n can be used as PCR primers.

Notation: For notational convenience, given an n-bit string x, we denote by $R_i(x)$ the DNA sequence encoding x at position i, i.e.

$$R_i(x) = B_i(x_1)\ S_i\ B_{i+1}(x_2)\ S_{i+1}\ \ldots S_{i+n-2}\ B_{i+n-1}(x_n)\ S_{i+n-1}$$

3.1 Operations on binary strings

Let T be a test tube containing DNA strands representing a set of binary strings. The encoding described above enables one to extract binary strings satisfying certain properties. For instance, to extract all strands in T representing binary strings whose i'th bit is 1, one simply extracts all strands containing the pattern $B_i(1)$ as described in Section 2.2.1. Observe that our encoding scheme ensures that a strand contains the pattern $B_i(1)$ if and only if it represents a strings whose i'th bit is 1. We denote this operation by

$$\text{Extract}(\ T\ ,\ x_i = 1)$$

It is possible to do even more in one extraction. To extract all strands in T who have a 1 in their i'th position and a 0 in their $i+1$'st position one can extract all strands containing the pattern

$$R_i(10) = B_i(1)\ S_i\ B_{i+1}(0)\ S_{i+1}$$

We denote this operation by Extract(T , $x_i x_{i+1} = 10$). Hence, we were able to perform an "and" operation on consecutive bits for the cost of one extract operation. Note that the fact that the bits are adjacent is crucial for this to work.

A more general extract operation is the following:

$$\text{Extract}(\ T\ ,\ x_i x_{i+1} x_{i+2} = 101 \text{ or } x_i x_{i+1} x_{i+2} = 001)$$

This can be done using two types of streptavidin beads at once. One bead will attract all strands containing the string $R_i(101)$ and the other will attract all strands containing $R_i(001)$. Hence, for the cost of one extract operation we were able to perform a combination of an "and" and an "or". For technical reasons it is only possible to perform this simultaneous "or" operation when the strings being extracted have the same length.

4 The DES circuit

We give a brief overview of the DES circuit. DES encrypts a 64 bit plain-text into a 64 bit cipher-text using a 56 bit key. We denote by DES(M, k) the DES encryption of the plain-text M using the key k.

As will be seen in the next sections, our goal is to run the DES circuit on a fixed 64 bit string M using all possible keys k. The circuit computing the function $f(k) = \mathrm{DES}(M, k)$ is shown in figure 2. The figure displays a convoluted version of the DES circuit since in our case the message to be encrypted is fixed and the key varies. Traditionally the circuit is drawn the other way around, i.e. the key is fixed and the message varies. For a more detailed description of the DES circuit see [5].

We now explain the various components of figure 2. The circuit is composed of 16 levels. We refer to each level as a round. The figure shows the first four rounds and the last round. The input to the circuit is the 56 bit key shown on the left. The high order 32 bits of M are denoted by M_h and the low order bits are denoted by M_l.

A P-box is a box which permutes the bits of its input. The box may be set to discard some of the input bits or to replicate some of them. The exact operation performed by each P-box in the figure is predetermined and can be found in [5]. For our purposes we note that the P-boxes are insignificant and may be ignored. The reason for this is that P-boxes simply change the order of the bits that arise during the computation. When going through a P-box there is no need to physically change the order of the bits in the DNA-strand. We can simply keep track in our mind which bits on the strand encode which bits of the computation.

An S-box is more complicated. An S-box takes 48 bits of input and outputs 32 bits. The S-box groups the 48 input bits into 8 groups of 6 bits each. For each group of 6 bits the S-box performs a table lookup and outputs 4 bits. This can be seen in figure 3. The 8 tables used can be found in [5]

To summarize, we see that each DES round involves a 48 bit Xor, an S-box and another 32 bit Xor. Thus, each round requires 80 Xor operations and 8 table lookups.

5 Plan of DES attack

We are now ready to explain our attack on DES. We begin by giving an abstract overview of how we intend to recover the required key k. We then go into the details specific to breaking DES.

Let $f : \{0,1\}^n \to \{0,1\}^m$ be some function. We intend to construct a solution of DNA strands which encode all pairs $[k, f(k)]$. More precisely, for each $k \in \{0,1\}^n$ the solution will contain at least one DNA molecule representing the binary string $k, f(k)$. Such a solution may be regarded as a lookup table for all the possible values of the function $f(k)$. We denote this solution by T_f. Recall that if $2^n < 10^{17}$ then the solution T_f will fit into a liter of water. This condition implies that n should be at most 57. Clearly this bound can be raised by using larger volumes.

Given the solution T_f it is a simple matter to invert the function f. By inverting the function we mean that given a value E one can find a k such that $f(k) = E$. Such a k can be recovered using the following two steps:

1. Extract from T_f all strands that contain the pattern $R_{n+1}(\ E\ )$. Notice that the extracted strands encode pairs of strings $[k, f(k)]$ where $f(k) = E$.

2. The value of k can now be obtained by reading the binary string written on any of the extracted strands. This can be done by sequencing any of the extracted strands.

This two step process will produce a string k such that $f(k) = E$. Moreover, both steps can be carried out within several hours of work in the lab. Thus, we see that given the DNA solution T_f one can quickly invert the function f. This is the key fact we use for breaking cryptographic schemes. Notice that here we made use of the fact that a DNA solution can store the entire lookup table for the

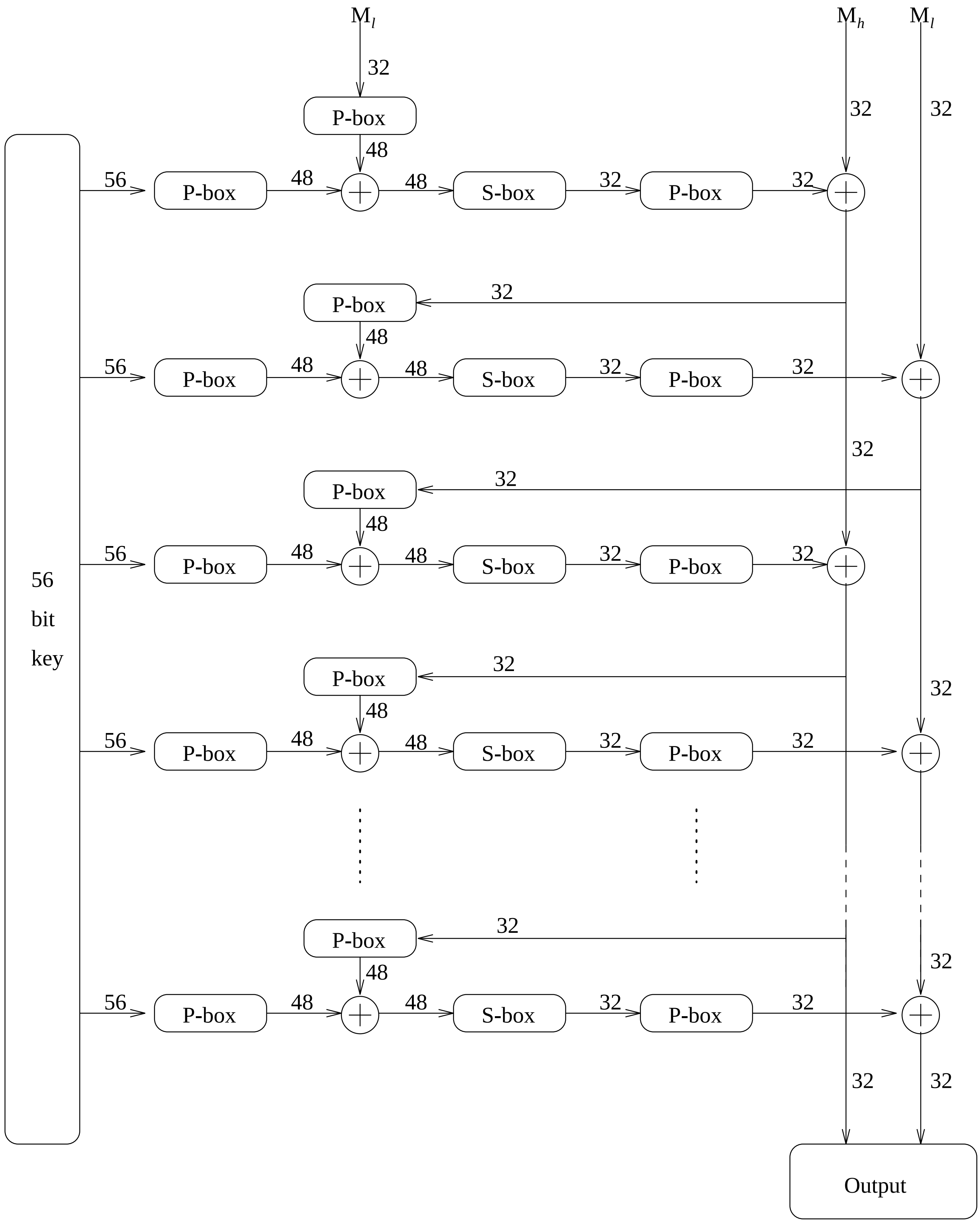

Figure 2: DES circuit

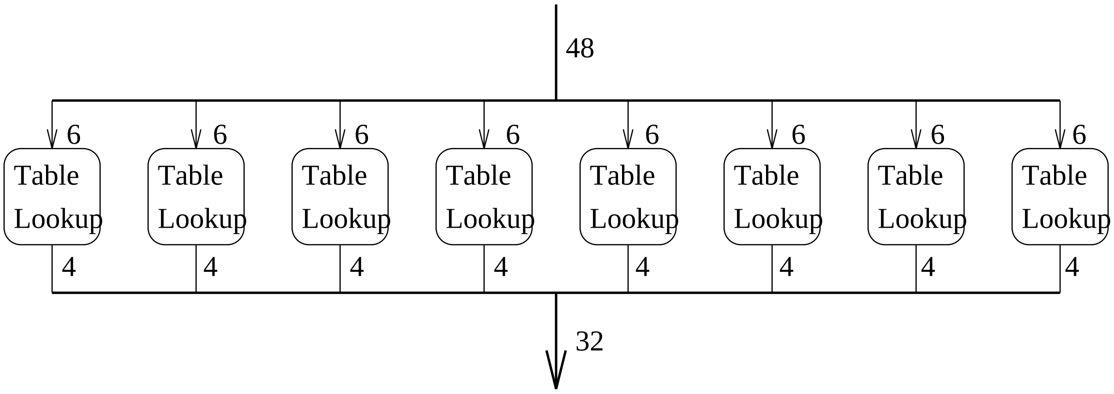

Figure 3: S-box circuit

function f. This is an important point since it shows that DNA can be used as an extremely dense storage medium.

The above method can be used by an eavesdropper to break DES. We call the eavesdropper Eve. Given a (plain-text, cipher-text) pair (M_0, E_0) Eve's objective is to find a key k_0 which maps the plain-text M_0 to the cipher-text E_0. A property of the DES circuit is that this key k_0 is likely to be unique. We define the following function:

$$g(k) = \mathrm{DES}(M_0, k)$$

The function $g(k)$ maps a 56 bit key to a 64 bit cipher-text. Eve's objective is to find a key k such that $g(k) = E$. In other words, Eve wishes to invert the function $g(k)$.

We intend to use the method outlined above to invert the function $g(k)$. The first step is to construct the solution T_g. Notice that the solution T_g depends on the plain-text M_0. We therefore choose to denote the solution T_g by $\mathrm{DES}(M_0)$. In the next section we show how to construct the solution $\mathrm{DES}(M_0)$. As we shall see, this step is laborious and requires about 4 months of work in the lab. Once the solution $\mathrm{DES}(M_0)$ has been constructed, Eve can quickly break the cryptosystem using the two step procedure above. Thus, the time it takes to break DES given an arbitrary (plain-text, cipher-text) pair is dominated by the time is takes to construct the solution $\mathrm{DES}(M_0)$. A crucial fact is that the solution $\mathrm{DES}(M_0)$ contains "only" 2^{56} strands. Since $2^{56} < 10^{17}$ we see that the entire computation can be carried out in one liter of water.

We now show how Eve can break DES in *one day* using chosen plain-text attack and some preprocessing. The idea is for Eve to choose a 64 bit message M_0 and construct the solution $\mathrm{DES}(M_0)$. She does this off-line as a preprocessing step. Once she has the solution $\mathrm{DES}(M_0)$ she can use it to break many DES cryptosystems in one day. To break a DES box she simply generates the pair (M_0, E_0) where E_0 is the encryption of her fixed message M_0. She can do this since by assumption she is allowed to encrypt a chosen plain-text. She now uses the solution $\mathrm{DES}(M_0)$ and the two step process above to find the key k mapping M_0 to E_0.

The point is that once Eve constructed the solution $\mathrm{DES}(M_0)$ she can use it break many cryptosystems very quickly and for very little cost. This procedure shows that a chemical company can spend the time to generate the solution $\mathrm{DES}(M_0)$ and then sell it to anyone who wants to quickly break a DES system.

6 Evaluating the DES circuit

In this section we describe how to generate the solution DES(M) for some given plain-text M. As we saw in the previous section, this solution is the key ingredient in the molecular attack on DES.

We give a quick outline of how to construct this solution and then describe the details. The process is carried out in three steps.

1. Construct an initial solution encoding all 56-bit strings. This solution contains all strands of the form

   $$\updownarrow S_0 \; R_1(k)$$

   for all $k \in \{0,1\}^{56}$.

2. Evaluate the gates of the DES circuit by tagging on their values. For instance, suppose the first gate in the DES circuit instructs us to Xor bit 3 and bit 7 of the key. We do this by appending the value of $k_3 \oplus k_7$ to each of the strands in the initial solution. Here k_i denotes the i'th bit of the key k. The resulting solution will contain strands of the form:

   $$\updownarrow S_0 \; R_1(k) \; B_{57}(k_3 \oplus k_7) \; S_{57}$$

   for all $k \in \{0,1\}^{56}$. This "tagging" operation can be carried out using the procedure discussed in section 2.2.3. Observe that before the tagging took place all strands contained the string S_{56} on one end.

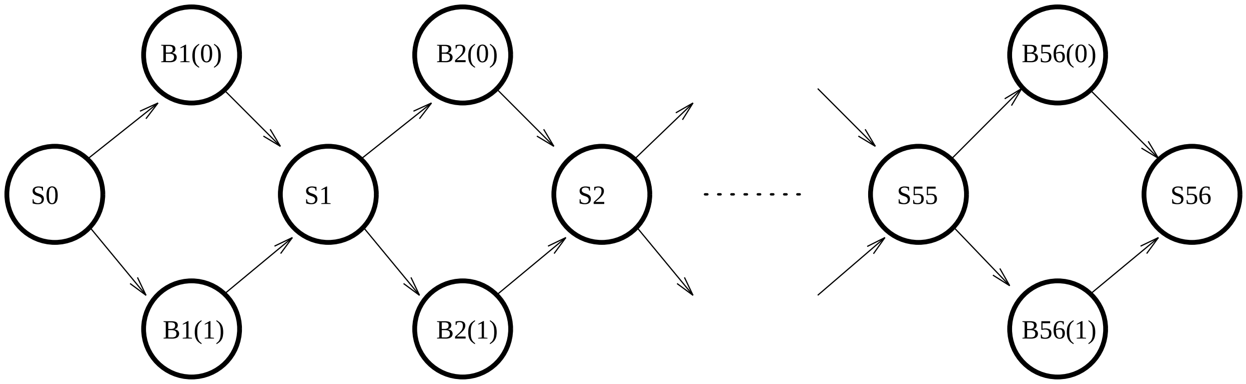

Figure 4: Initialization graph

3. We evaluate the gates in the circuit one by one. For each gate we append the output of the gate to all strands in the solution. When we are done, all strands in the solution look like:

$$\updownarrow S_0\ R_1(k)\ \ R_{57}(\ I\ )\ \ R_r(\ \mathrm{DES}(M,k)\ )$$

where I is a string of bits corresponding to the values generated by internal gates in the circuit. This final solution is the one we set out to construct since the strands in the solution encode all pairs $[k, \mathrm{DES}(M,k)]$.

6.1 Constructing the initial soup

To construct the initial solution of 2^{56} DNA strands corresponding to all possible DES keys we use a procedure similar to the one described in [9]. The basic idea is to use the graph G shown in figure 4. Each path from S_0 to S_{56} corresponds to one DES key. To create a soup of DNA strands which correspond to all paths in G we use the encoding of [1]. We follow the following steps:

1. In Section 3 we assigned a unique oligo of length 30 to each vertex in the graph. Denote the oligo assigned to vertex V by $V^l V^r$ where V^l represents the 15 left bases and V^r represents the 15 right bases.

2. For each vertex v create a test tube containing the strands $\uparrow V^l V^r$. For each directed edge (u, v) create a test tube containing the strands $\downarrow U^r V^l$. To handle the end points, create two test tubes containing $\uparrow S_0^l$ and $\downarrow S_{56}^r$.

3. Mix all the test tubes and allow the strands to anneal and ligate. Then extract from the resulting mixture all the strands that contain the words S_0. Call the set of extracted strands A. From the solution A extract all strands that contain the word S_{56}.

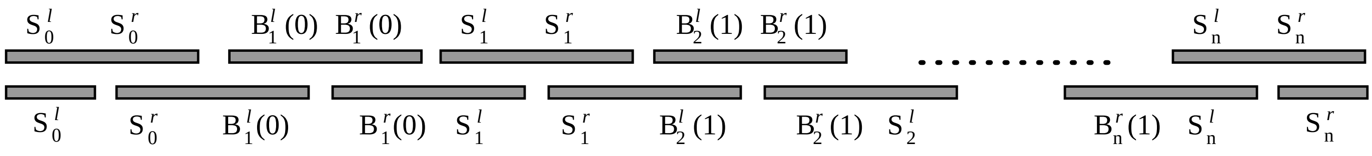

Figure 5: A path in the graph

It is not difficult to see that at the end of step 2 we get a mixture that corresponds to various paths in the graph. Figure 5 shows an example of a DNA strand which is created when all the test tubes are mixed. At the end of step 3 the resulting strands will correspond to all paths from S_0 to S_{56}. The strand of Figure 5 is an example of such a strand. The strand in the figure encodes an n-bit binary string whose first two bits are 01 and whose last bit is 1.

6.2 Evaluating Xor gates

It remains to show how to evaluate the various components of the DES circuits. There are only two types of gates in the DES circuit: an Xor gate and a lookup table (used in the S-boxes). We begin by explaining how to evaluate an Xor gate.

Suppose we wish to evaluate the Xor of the i'th and j'th bits of the strings in the current solution T. This is done by appending the value $x_i \oplus x_j$ to the strands representing x. The first step towards performing this operation is to separate the solution T into two solutions T_0, T_1 where T_0 contains those strands in T satisfying $x_i \oplus x_j = 0$ and T_1 contain those satisfying $x_i \oplus x_j = 1$. We then tag each of T_0, T_1 with the appropriate value.

We can write $x_i \oplus x_j$ as

$$x_i \oplus x_j = (x_i + x_j)(\bar{x}_i + \bar{x}_j) \quad ; \quad \overline{x_i \oplus x_j} = (\bar{x}_i + x_j)(x_i + \bar{x}_j)$$

This leads to the following procedure for generating T_0, T_1:

1. Set $T^{11} = \text{extract}(T\ ,\ x_i = 1 \text{ or } x_j = 1)$ and $T^{01} = \text{extract}(T\ ,\ x_i = 0 \text{ or } x_j = 1)$.
2. Set $T_1 = \text{extract}(T^{11}\ ,\ x_i = 0 \text{ or } x_j = 0)$ and $T_0 = \text{extract}(T^{01}\ ,\ x_i = 1 \text{ or } x_j = 0)$.

This procedure requires 4 extract operations. However, if we allow two extract operations to be done in parallel, then the procedure can be carried out in only two steps.

To summarize, one Xor gate can be evaluated in time equivalent to two extraction steps and one tagging step. If we allow more extract steps to be carried out in parallel, then we can evaluate more than one Xor gate at a time. It can be shown that by allowing 32 extract operation to be carried out in parallel, it is possible to evaluate three Xor gates in two extraction steps and one tagging step.

6.3 Evaluating a lookup table

A lookup table is a function $f : \{0,1\}^6 \to \{0,1\}^4$ which maps 6 bit strings to 4 bit strings. The function is given as a table of values. In the DES circuit it is possible to arrange things so that the six bits to be looked up are always consecutive bits. This can be arranged by appending bits in the "right" order. In Section 3.1 we observed that when the six bits are consecutive it is possible to extract all strands that contain a specific six bit pattern in *one* extract step.

Suppose the bits to be looked up are in position i to $i+5$. We wish to append to every strand the four bit string $f(x)$ where x is the six bit pattern encoded on the strand in position i to $i+5$. This is done by separating the solution T into 16 solutions $T_0, \ldots, T_{15}$ according to the value to be appended. We then tag each T_j by the 4-bit string j.

For example, suppose the function f maps the two six bit strings a and b to 0 and they are the only ones mapped by f to 0. Then the solution T_0 can be extracted from T in one extract step by setting:

$$T_0 \leftarrow \text{Extract}(\ T\ ,\ x_i \ldots x_{i+5} = a \ \text{ or } \ x_i \ldots x_{i+5} = b)$$

The other solutions $T_1, \ldots, T_{15}$ can be extracted in a similar way. This shows that a lookup table can be evaluated using 16 extractions and 16 tagging steps. Furthermore, these operation can be done in parallel. Hence, if we allow 16 extractions to be done at once, then table look up can be done in time which is approximately equivalent to one extract operation and one tag operation.

The separation step requires 64 different types of beads corresponding to all 6 bit combinations. These beads can be prepared using the same method used to construct the initial soup of all 56 bit strings, i.e. by constructing all paths in a graph similar to Figure 4 which is made up of only 6 diamonds. As we will see in Section 6.4 the same set of 64 different beads can be used for all rounds of the DES circuit. Hence, they need only be prepared once at the beginning of the computation. The same method can be used to generate the 16 bit patterns that are appended to the strands.

6.4 Shortening the strands

The process of continually appending values to the strands would make the strands very long by the time we complete the DES circuit. It is possible to shorten the strands by throwing away segments which are no longer needed. This can be done at the end of every round. The only values that have to be kept on the strands are the 56 bit key and the 64 bits that propagate out of the previous round. By appending bits in a clever order it is possible to arrange things so that the redundant bits are always located at the ends of the strands. Using restriction enzymes it is possible to cut these redundant bits off the strands and remove them from the solution.

Since the B and S strings used by the junk portion are no longer present in the solution they may be reused. For instance, if the 57'th bit was cut away from all strands then the strings $B_{57}(1)$, $B_{57}(0)$ and S_{57} can be reused in the next round. Specifically, the same set of 64 beads used in the table lookup step can be used in all rounds.

6.5 Summary

We briefly calculate the number of steps required to construct the solution DES(M) using the methods sketched thus far. First, observe that all Xors on the first round may be ignored. Similarly, the right 32 Xors on the second round may also be ignored. These Xors are computing an Xor with the message bits. Since the message bits are fixed, these Xors are either a no-op (i.e. no operation performed) or a negation. A negation only requires that we make a mental note remembering that the string representing 1 is now representing 0 and vice versa. Thus, the first 48+32+32=112 Xors may be disregarded.

Suppose we allow 32 parallel extractions per step. We regard such a step as a *parallel extraction step.* In Section 6.2 we noted that one can evaluate 3 Xors in two parallel extract steps. In Section 6.3 we saw that one can evaluate a lookup table in one parallel extract step.

There is a total of 16*8=128 table lookups in the DES circuit. As noted above, in all but the first round we need to evaluate the left 48 Xors. By evaluating 3 Xors in two parallel extract steps this takes $2 * 15 * 48/3 = 480$ parallel extract steps. Similarly, in all but the first two rounds we must evaluate the right 32 Xors. This takes $2 * 14 * \lceil 32/3 \rceil = 308$ parallel extract steps. Thus, the total number of parallel extract steps is

$$480 + 308 + 128 = 916$$

7 Conclusions

The objective of this paper was to provide a concrete example of a hard problem which can be solved on a molecular computer. Our techniques also provide an example of how a molecular computer might be programmed. We emphasized the fact that boolean gates are evaluated by tagging on their value. Furthermore we saw how to evaluate Xor gates and lookup tables.

We briefly summarize our results regarding breaking DES. We showed that assuming one can perform 32 extractions per step, it is possible to break DES in 916 steps. Our figure of 32 extractions per step is a reasonable one. The reason is that most biological apparatus, such as PCR machines, act on dozens of solutions at once. Recently Kaplan [7] repeated Adleman's experiment. Kaplan reports that an extraction step can be carried out in roughly an hour. Thus we can approximately claim that 10 extraction steps per day is a reasonable figure. Under such an assumption we see that one can break DES in 4 months. We point out that our procedure for breaking DES has not been carried out in the lab and therefore our 4 months figure should only be regarded as a rough estimate.

The attack on DES described in this paper is very general. It can be applied to any cryptosystem which uses keys of length less than 64 bits provided the encryption circuit is not too big. This leads to an interesting consequence of molecular computing: cryptosystems using keys shorter than 64 bits are insecure. The mere possibility that a molecular computer can break such cryptosystems is sufficient to invalidate them.

Currently the future of molecular computers is unclear. It is possible that in the future molecular computers will be the clear choice for performing massively parallel computations. However, there are still many technical difficulties to overcome before this becomes a reality. We hope that this paper helps to demonstrate that molecular computing is a technology worth pursuing.

References

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

[2] L. Adleman, P. Rothemund, S. Roweis, E. Winfree, "On Applying Molecular Computation to the Data Encryption Standard", in proceedings of second annual conference on DNA based computers, Princeton, 1996.

[3] D. Boneh, C. Dunworth, R. Lipton, J. Sgall, "On Computational Power of DNA", Princeton CS Tech-Report CS-TR-499-95, `ftp://ftp.cs.princeton.edu/reports/1995/499.ps.Z`. To appear in the special issue of Discrete Applied Mathematics on Computational Molecular Biology.

[4] D. Boneh, R. Lipton, "Making DNA Computers Error Resistant", Princeton CS Tech-Report CS-TR-491-95, `ftp://ftp.cs.princeton.edu/reports/1995/491.ps.Z` in proceedings of second annual conference on DNA based computers, Princeton, 1996.

[5] National Bureau of Standards, "Data Encryption Standard", U.S. department of commerce, FIPS, pub. 46, January 1977.

[6] E. Biham, A. Shamir, "Differential Cryptanalysis of the Full 16-round DES", In Proceedings of Crypto 1992, pp. 487–496.

[7] P. Kaplan, "Molecular Computation: Adleman's experiment repeated", NEC Research Institute, Princeton, NJ, Technical Report # 95-120.

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

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

[10] M. Matsui, "The first experimental cryptanalysis of the Data Encryption Standard", In Proceedings Crypto 1994, pp. 1–11.

[11] P. Rothemund, "A DNA and restriction enzyme implementation of Turing machines", in DNA Based Computers, volume 27 of DIMACS: Series in Discrete Mathematics and Theoretical Computer Science.

[12] M. Wiener, "Efficient DES Key Search", TR-244, School of Computer Science, Carleton University, May 1994.