The Power of Surface-Based DNA Computation (Extended Abstract)

Weiping Cai, Anne E. Condon, Robert M. Corn, Elton Glaser,
Zhengdong Fei, Tony Frutos, Zhen Guo, Max G. Lagally,
Qinghua Liu, Lloyd M. Smith, Andrew Thiel

University of Wisconsin
Madison, WI 57306 USA

Abstract

A new model of DNA computation that is based on surface chemistry is studied. Such computations involve the manipulation of DNA strands that are immobilized on a surface, rather than in solution as in the work of Adleman. Surface-based chemistry has been a critical technology in many recent advances in biochemistry and offers several advantages over solution-based chemistry, including simplified handling of samples and elimination of loss of strands, which reduce error in the computation.

The main contribution of this paper is in showing that in principle, surface-based DNA chemistry can efficiently support general circuit computation on many inputs in parallel. To do this, an abstract model of computation that allows parallel manipulation of binary inputs is described. It is then shown that this model can be implemented by encoding inputs as DNA strands and repeatedly modifying the strands in parallel on a surface, using the chemical processes of hybridization, exonuclease degradation, polymerase extension, and ligation. Thirdly, it is shown that the model supports efficient circuit simulation in the following sense: exactly those inputs that satisfy a circuit can be isolated and the number of parallel operations needed to do this is proportional to the size of the circuit. Finally, results are presented on the power of the model when another resource of DNA computation is limited, namely strand length.

*Corn, Fei, Frutos, Guo, Liu, Smith and Thiel are in the Chemistry Department, Condon and Glaser are in the Computer Sciences Department and Cai and Lagally are in the Materials Sciences Department. Email address for further communication: condon@cs.wisc.edu. Supported by NSF grant numbers CCR-9628814 and CCR-9613799.

1 Introduction

Following Adleman [1], Lipton [7], and others, we recently proposed a new DNA computing scheme based on surface chemistry [8]. In this paper, we define an abstract model of surface-based DNA computation and describe the power and limitations of this model.

In contrast with Adleman's solution-based experiment, surface-based DNA computations manipulate DNA strands that are immobilized on a surface using chemical linkers. This means that a key operation used in solution-based DNA computations, that of selectively separating strands into separate test tubes, cannot be performed. Also the number of DNA strands involved in the computation is limited since the strands are restricted to two rather than three dimensions. (The number of strands that can fit on a 1 cm^2 planar surface is roughly 10^{12}.) Nevertheless, it is our premise that surface-based chemistry will be important to advances in DNA computation for the following reasons.

First, surface-based chemistry has been a critical technology in recent advances in biochemistry, including protein sequencing, DNA synthesis, and peptide synthesis [12]. Second, handling of samples is simpler and more readily automated when the samples are immobilized on surfaces rather than in solution, loss of strands is effectively eliminated and a much greater degree of control is obtainable in each chemical "operation." This is important since errors, particularly in separating strands into distinct test tubes, is a primary obstacle to Adleman's approach. Third, the work of the company Affymetrix on

DNA synthesis on "addressed" surfaces [11] can be viewed as a restricted form of computation. Our proposed DNA computing method is complementary to Affymetrix arrays in that the ability to address strands spatially is lost but the scale is much greater. Finally, experiments on surfaces will provide an ideal means to develop chemistry, algorithms and understanding of errors that will be useful in further, larger-scale DNA computations that may not necessarily be wholly based on surfaces.

For all of these reasons, we believe that it is important to understand how general-purpose computation can be performed on surfaces. Towards this end, we introduce an abstract model of computation that can be implemented using standard surface chemistry. Roughly, the operations in the model allow strands to be marked or selected depending on their value; bits to be appended to the free (non-immobilized) end of selected strands; destruction of unmarked strands; and erasure of appended bits from strands. We then show that surface-based DNA chemistry efficiently supports general circuit computation on many inputs in parallel: Given a circuit that accepts a language $L_n \subseteq \{0,1\}^n$, there is a surface-based DNA algorithm that, given a set S of binary inputs that are represented as DNA strands, identifies exactly those inputs in S that satisfy the circuit and destroys the rest. The values of gates are appended in parallel to strands until the output is computed. The number of operations needed to do this is proportional to the size of the circuit.

In addition to keeping the number of operations to a minimum in a DNA computation, it is also desirable to keep the maximum strand length small. One reason for this is that long strands are more likely to form unwanted secondary structures that cause errors in the computation. Therefore, we prove results about the power of surface-based computation when modification of strands by appending information onto them is limited.

In Section 1.1 we present our abstract model of surface-based DNA computation. We then summarize our results on the power of the model in Section 1.2. The chemical basis for the abstract model is given in Section 2. Proofs of the results are given in Section 3.

1.1 Abstract Model

The following surface-based model is motivated by several goals. It should be built on chemical processes that are reasonably well understood and reliable. It should be conceptually simple, yet the operations should exploit the chemical processes in as general a way as possible. This is because it is important to keep the number of operations to a minimum in a DNA computation; the more general the operation set, the fewer operations are needed in a computation.

Operations are performed on a multiset S of strands, where each strand is a sequence of bits. Several practical and strategic considerations detailed in Section 2 motivate the following word-based structure for organizing bits within strands. Each strand is a finite sequence of short, variable-length binary words. The length of a word is the number of bits in that word. Each word in any strand has an index; the indices of words on the same strand must be distinct. Over all strands, words with a common index i have the same length. The value of a word is the vector of values of the bits within that word. For simplicity we assume that all strands in the initial set (i.e. before any operation is performed) have the same number, say n, of bits which are organized into words with the same indices. We now list the operations that can be applied to a set S of strands.

mark(constraint): all strands satisfying the constraint are identified as marked. A constraint specifies, for some index i, a set of values for a subset of the bits in a word with index i. A strand satisfies this constraint if and only if there is a word with index i on the strand and the value of this word agrees with the values of the bits specified in the constraint. The mark operation with no constraint is also allowable; in this case all strands are marked.

unmark: unmark all marked strands.

destroy-unmarked: unmarked strands are removed from the set S.

append-marked(new-i, new-word): a word with index new-i and value new-word is appended to all marked strands. When this and the following append-unmarked operation are used in an algorithm, the algorithm must ensure that no strand has more than one word with the same index.

append-unmarked(high-i, high-word, new-i, new-word): a word with index new-i and value new-word is appended to all unmarked strands in which the index of the highest-numbered word is high-i and its value is high-word.

erase: all words that have been appended to strands since the start of the computation are erased leaving just the initial strands, all of which are unmarked. The motivation for including this operation is to keep strands short.

We note that by using the mark operation, strands can be selectively marked for further "processing" such as append or destroy. The mark operation replaces the separate (or extract) operation of Adleman and Lipton, which divides strands into separate test tubes for further processing. The destroy-unmarked operation is very similar to an operation of Amos et al. [3] although their model is solution-based. There is no read-out operation in the above set. In this paper we ignore readout and focus on what sets can be recognized, as described in the next section. The problem of read-out is essentially the same for surface-based DNA computation as for solution-based DNA computation.

1.2 Power of Model

A DNA algorithm A $(= A_n)$ is a sequence of operations applied to an initial multiset S of n-bit strands. The output of the algorithm, denoted by $A_n(S)$, is the multiset of strands on the surface at the end of the algorithm. We say that A_n recognizes, or accepts, a subset L_n of $\{0,1\}^n$ if and only if $A_n(\{0,1\}^n) = L_n$. To characterize the power of DNA algorithms, we adopt standard conventions used for circuits. Consider a (nonuniform) infinite family of DNA algorithms $\{A_n\}$, one algorithm per input size. We say that this family accepts language L if and only if the language accepted by A_n is $L \cap \{0,1\}^n$.

We consider the following resources of a DNA algorithm: (i) the number of operations; (ii) the maximum number of bits, or alternatively of words, appended to a strand during the computation; and (iii) the types of operations. We distinguish between computations that use the append operation and those that do not. This is because computations that use append rely on more complicated chemistry and result in longer strands.

Let Surface-Time($t(n)$) be the set of languages accepted by a family of DNA algorithms with $O(t(n))$ operations on n-bit strands. The first result is that Surface-Time($t(n)$) is exactly the set of languages that are accepted by a family of bounded fan-in circuits with $O(t(n))$ gates. Let Circuit-Size($t(n)$) be the latter set. The constants in the O-notation in the following theorems are made explicit in the proofs.

Theorem 1 *Surface-Time*$(t(n))$ = *Circuit-Size*$(t(n))$.

This theorem is analogous to the theorem of Boneh et al. [4] that the Circuit SAT problem can be solved using a solution-based model of DNA computation.

We are also interested in computations of languages when the append operation is not allowed. In this case, the erase operation is also unnecessary. We let Restricted-Surface-Time($t(n)$) be the set of languages recognized by such algorithms with $O(t(n))$ operations. A language in this class can be expressed as the set of satisfying assignments of a boolean formula of the following type. The variables of the formula are partitioned into words with the number of variables per word bounded by the same constant that bounds the number of bits per DNA word. The formula is represented as a depth-three tree with the output gate (root) at level 0 and the n input variables and their negations labeling the leaves at level 3. The root is an unbounded fan-in **and**-gate. The gates at level 1 are unbounded fan-in **or**-gates. The gates at level 2 are bounded fan-in **and**-gates with the additional restriction that all inputs to an **and**-gate at level 2 must be variables (or their complements) from a single word. Call such a formula an extended-sat formula. Let Extended-Sat($t(n)$) be the set of languages recognized by such formulas with n variables and $O(t(n))$ gates.

Theorem 2 *Restricted-Surface-Time*$(t(n))$ = *Extended-Sat*$(t(n))$.

Let Surface-Time,Length$(t(n), l(n))$ be the set of languages that are computable by a family of DNA algorithms with $O(t(n))$ operations in which the maximum number of bits that are appended to any initial strand is $O(l(n))$ throughout the computation. From Theorem 1 it follows immediately that Circuit-Size($t(n)$) = Surface-Time,Length$(t(n), t(n))$. To state our result about the languages in Surface-Time,Length$(t(n), l(n))$ for general $l(n)$, we need to define a new type of circuit, which we call a *blocked* circuit. All gates but one of such a circuit are organized into blocks; each block has a single output gate. The output gates of the blocks are input to a single unbounded fan-in **and**-gate, which is the output of the circuit. We let Circuit-Size,Block$(t(n), l(n))$ be the set of languages accepted by blocked circuits with $O(t(n))$ gates and $O(l(n))$ gates per block.

Theorem 3 *Circuit-Size,Block*$(t(n), l(n)) \subseteq$ *Surface-Time,Length*$(t(n), l(n))$.

The above results can readily be extended to function computation, defined as follows. We say that a family $\{A_n\}$ of algorithms computes a function f if $A_n(\{0,1\}^n)$ is the set of strands of the form $xyf(x)$, where $x \in \{0,1\}^n$ and y is an arbitrary strand.

2 Chemical Realization of Model

We now describe how the model of Section 1.1 can be implemented using surface-based DNA chemistry. Preliminary work on implementing the mark and destroy operations on single-word strands is already underway [8].

2.1 Set Initialization; Word Design

A DNA strand, or oligonucleotide, can be thought of as a string over the alphabet (i.e. set of bases or nucleotides) $\{A, C, G, T\}$ with chemically distinct ends known as the $3'$ and $5'$ ends. A DNA oligonucleotide is used to encode the value of a word plus the index of that word. A *strand* in our abstract model is simply the concatenation of multiple words with distinct indices. Also a common sequence of bases known as a primer is placed at the $3'$ end of each strand to enable readout using the polymerase chain reaction (PCR).

To encode binary truth assignments for (up to 70) variables, Adleman and Lipton proposed that one-bit words be used, with 20 bases per word to identify the bit value and the variable (or word) index [2, 7]. The exact encoding of words was not specified. Each possible word can be synthesized using standard solid phase DNA synthesis in which a large quantity of a desired DNA molecule is built up nucleotide by nucleotide on a support particle in sequential coupling steps. For example, a support with the nucleotide "A" attached may have the "A" reacted with a "C" to form a string of length 2 (known as a dimer), washed and the "C" coupled with "G" to form a string of length 3 (a trimer), and so on. Adleman [2] described how words can be concatenated together using a chemical process called ligation so that all 2^n n-bit strands can be created in n ligation steps.

In our preliminary experimental work on an initial multiset $S = \{0, 1\}^5$ [8], we are using one base per bit, plus labels on either end. This can extended to multiple words as follows. The word index is encoded using the labels and between the labels the bases A or T are used in half of the bit positions to represent 0 and 1 respectively, and C or G are used in the remaining positions to represent 0 and 1 respectively. The DNA synthesis scheme described above can easily be adapted for parallel synthesis of multiple-word strands when this method of encoding is used. Namely, a mixture of two nucleotides is used at coupling steps that correspond to bits of a word and just one nucleotide is used at coupling steps that correspond to the label of a word. For example, if two nucleotides are used together in five coupling steps, 32 different molecules are made and are present on the support. In our experimental work, an Applied Biosystems DNA synthesizer is used to generate the initial set; with this system a set of strands with 50 bases can be generated in an afternoon for about $100. In addition to the possibility of parallel synthesis of words, this method has the advantage of high information density, i.e. bits per base. However words with the same index that differ in only one bit necessarily differ in only a single base, which makes implementation of a reliable mark operation more challenging as we explain later.

Schemes intermediate between the two already described are also possible in which there are multiple bits per word, but within a word there may be more than a single base per bit. However using the synthesis scheme described above in which single nucleotides are coupled at each step, parallel synthesis of words is no longer possible. To generate the set of all possible n-bit strands with w bits per word and greater than one base mismatch between any pair of words, $2^w n/w$ synthesis steps plus n/w ligation steps are required.

Attachment chemistry describes the molecules at the interface of the surface and the oligonucleotides to be attached to the surface. Both the surface and one end of the oligonucleotides are specially prepared to enable this attachment. A good attachment chemistry ensures that the properly prepared oligonucleotides are immobilized to the surface at a high density, and that other oligonucleotides exposed to the surface later (for example during hybridization) do not bind non-specifically to the surface. In our preliminary experiments on single-word strands [8], we use a glass surface and an attachment chemistry developed in the Smith laboratory [5] to attach oligonucleotides at the $5'$ end. The glass surface is modified with amino-reactive isothiocyanate functionalities in a multistep process. For multi-word strands it is necessary to attach oligonucleotides to the surface at the $3'$ end, in order to use polymerase extension in the mark operation as described later. A different chemistry can be used to do this.

2.2 Operations

We now describe how each of the operations of our abstract model can be realized on surfaces. All operations are based on standard chemical processes, namely hybridization, polymerase extension, ligation, and exonuclease degradation.

mark: Strands are marked simply by making them double-stranded at the free end. Under suitable con-

ditions (such as temperature and salt concentration), single-stranded DNA *hybridizes*, or anneals, to form a double-stranded DNA molecule, or duplex, with its Watson-Crick complement. In this duplex, complementary pairs of bases form a bond. The Watson-Crick complement of a DNA strand is the sequence obtained by replacing each A with a T and vice versa, similarly replacing each C with a G and vice versa, and with the distinct chemical ends in the opposite order. For example the Watson-Crick complement of $(5')$ACCTG$(3')$ is $(3')$TGGAC$(5')$. We begin by describing how the mark operation works in the case that all strands have a single word and then extend this to the case of multiple-word strands.

Single-word strands: First the set of DNA oligonucleotides F that are Watson-Crick complementary to the strands that satisfy the constraint are synthesized. If there are w bits in the word and j of these are constrained then there are 2^{w-j} distinct strings in F. Each of these hybridizes to its complement on the surface (if present). GC content has a very strong effect upon hybridization stability and hence upon hybridization conditions; hence if more than one bit is stored in a word, the GC content must be kept constant over all values of this word.

Hybridization discrimination refers to the degree to which only perfect matches occur in a hybridization reaction involving multiple strands. The fewer matches between two strands (or substrands), the less stable the hybridization between them. If one base is used to encode one bit in a word then single-base mismatch discrimination is needed to properly mark words and this is challenging to do successfully. This is why it may be preferable to use more than one base per bit when encoding words. Mir [10] discusses this further.

Multiple-word strands: In this case the method used for the single-word strands is combined with polymerase extension. First, as in the previous case, a multiset F of DNA oligonucleotides that are Watson-Crick complementary to the constrained word value are synthesized and annealed to the oligonucleotides to be marked. The word indices ensure that annealing takes place at the proper words.

After the hybridization step, any strand that satisfies the constraint will have a word annealed to it. This annealed word is used as a primer from which to initiate DNA synthesis. DNA polymerase adds nucleotides to the $3'$ hydroxyl (i.e. the $3'$ end) of the primer polynucleotide, copying the complementary sequence to the end. Thus marked strands are doubly-stranded from the constrained word to the free terminus. The mark operation can also be done with no constraints, that is, all unmarked strands can be marked using polymerase extension starting from the Watson-Crick complement of the initial primer that is common to every strand. The polymerase extension step is in principle linear in the length of a strand since it involves processing the strand base by base. However the process is quite fast in practice.

unmark: This is done simply by washing the surface in distilled water and raising the temperature if necessary. In the absence of salt, which stabilizes the double-stranded pairs, the complementary strands of DNA denature from the oligonucleotides on the surface and are washed away, leaving only the original single-stranded DNA attached to the surface.

destroy-unmarked: Single-stranded DNA molecules may be destroyed using enzymes known as exonucleases, which chew up DNA molecules from the end in. In fact exonucleases exist with specificity for either the single-stranded or double-stranded form. For example the enzyme Exonuclease I may be used to destroy single-stranded oligonucleotides.

append-marked(new-i, new-word): Since marked strands are double-stranded at the free terminus, the append operation can be implemented via ligation at the free terminus. The DNA polymerase employed in the mark operation will affect the nature of the free terminus of the marked strands. DNA polymerases which contain a $3' \rightarrow 5'$ proofreading activity (most naturally occurring procaryotic DNA polymerases do possess this activity) yield a blunt end. In this case blunt-end ligation can be used to implement the append operation. Ligation will only occur at the duplex and not at (unmarked) single-stranded oligonucleotides. If the $3' \rightarrow 5'$ proofreading activity is removed (such engineered polymerases are commercially available) they show a new non-templated $3'$ adenylation activity. That is, the base "A" is added to the $3'$ hydroxyl of the duplex creating a one base overhang, or "sticky end". A cloning vector developed in the Smith laboratory some years ago exploited this non-templated adenylation activity to facilitate the cloning of PCR products (TA cloning [9]). Ligation to "sticky ends" (using a complementary one base T overhang on the word to be ligated to the surface-bound duplex) is substantially more efficient than blunt-end ligation. Hence, in the case of a mark followed by an append with no destroy operation intervening, sticky-end ligation can be used.

append-unmarked(high-i, high-word, new-i, new-

word): This can be done using hybridization of a splint oligonucleotide followed by ligation, exactly as in Adleman's experiment [1].

erase: This operation is implemented using a restriction enzyme which recognizes a short sequence of bases called a restriction site and cuts the strand at that site when the sequence is double-stranded. The restriction enzyme may have a 4, 6, or 8 base pair recognition sequence and may leave a blunt end, 3' overhang, or 5' overhang. The restriction site may be included in all appended words or may only be included in the first appended word. It must be chosen to have a recognition sequence not represented in the combinatorial word structure. For example the restriction enzyme Msp I recognizes the 4 base-pair sequence $(5')$CCGG$(3')$, when annealed to its complement $(3')$GGCC$(5')$ [6]. To ensure that the operation is performed correctly, an unmark operation can first be performed, then the restriction site can be made double-stranded on all strands via hybridization, followed by use of the restriction enzyme.

3 Proofs of Results

Theorem 1 *Circuit-Size*$(t(n))$ = *Surface-Time*$(t(n))$.

Proof: Let C be a bounded-degree circuit with n inputs and t gates. Assume that all gates of C are either **or**-gates or **and**-gates and the n inputs are available both in negated and unnegated form.

There are $k+4$ operations in the DNA algorithm per gate of fan-in k, plus five additional operations, two at the start and three at the end. Initially a word with a single bit "0" is appended to all strands using the mark operation with no constraint and the append-marked operation in which a new word with single bit "0" is appended. Then for each strand in parallel, the value of each gate of the circuit, given this strand as input, is computed in topological order and is appended to the strand.

If the gate is an **and**-gate with fan-in k then k mark operations mark those strands that set the **and**-gate to false. An append-marked operation appends a word containing the bit "0", which is the value of the gate for marked strands. Two append-unmarked operations append a word containing the bit "1" to unmarked strands. Two operations are needed because the append-unmarked operation requires knowledge of the last word on a strand. This can only have two possible values since each appended word contains just one bit. (This is the reason for the initial append operation.) Finally the unmark operation is done.

If the gate is an **or**-gate with fan-in k then k mark operations mark exactly those strands that set the gate to true. An append-marked operation appends a word containing the bit "1", which is the value of the gate for marked strands. Two append-unmarked operations append a word containing the bit "0" to unmarked strands. Finally the unmark operation is done.

The last three instructions of the DNA algorithm are: mark all of those strands for which the output of the circuit is "1," destroy-unmarked strands, and erase. The strands remaining on the surface are exactly those from the initial set that satisfy the circuit.

We next show how a DNA algorithm can efficiently be simulated by a circuit. Let A be a DNA algorithm with n inputs and t operations. We describe how to construct a circuit that accepts the language $A(\{0,1\}^n)$.

For convenience this circuit has two gates with fixed values 0 and 1. The remaining gates are added to the circuit for each operation in order that the operations appear in the algorithm A. To guide the construction of the circuit, some gates already added to the circuit are named (possibly with more than one name for a gate). Consider these names to be variables whose values (gates) are a function of the number of operations already processed. First one gate is named *marked*. Throughout the construction the following invariant is maintained: a strand in the initial multiset S is marked after t operations of the algorithm A if and only if when this strand is the input to the circuit the gate named *marked* has value "1" after t operations have been processed. For short we say that *marked* is a gate that records whether the input strand is currently marked. Initially the *marked* gate is the gate with fixed value 0. Second, for each distinct word index i used in the DNA algorithm one gate is named *present*(i). This gate records whether the word with index i is currently appended to the input strand. Initially *present*(i) is the gate with fixed value 1 if i is one of the initial words on the strand; else *present*(i) is the gate with fixed value 0. Third, for each index i that is used by the algorithm, for each bit position j of word i, *current-value*(i,j) is a gate which records the current value of the jth bit of the word with index i, if this word is present on the strand. If the word is not present this gate has arbitrary value. Initially the n input gates are named by the corresponding *current-value* variables for the input bits and the remaining variables name arbitrary gates. Finally one gate is named *output*; initially this is the fixed gate with value 1. On completion of the construction, *output* will refer to

a gate whose value is 1 if and only if the input strand is not destroyed by the DNA algorithm.

We can now describe how the circuit is constructed. Suppose that the first $t-1$ operations in the algorithm have been processed and consider the tth operation. Suppose that this is a mark operation. If its constraint refers to the word indexed i then a subcircuit is added to determine if word i is currently present on the strand, and if it is present, the subcircuit determines whether the current value of this word matches the constraint. The inputs to this subcircuit are the gates *present*(i) and *current-value*(i,j) for each bit j of word i. The output of this subcircuit is 1 if and only if the word with index i is present on the strand and the strand satisfies the constraint on word i. An **or**-gate g receives as input the output of this subcircuit and also the gate currently named *marked* (i.e. the gate referred to by *marked* after $t-1$ operations have been processed). Finally the *marked* variable is updated to refer to gate g.

Next suppose that the tth operation is the append-marked(new-i, new-word) operation. In this case a new **or**-gate g is added that takes as input the gate named *marked* and the gate named *present*(new-i). Then the variable *present*(new-i) is updated to refer to this new gate g. Also a subcircuit is added that appropriately updates the value of word new-i. This circuit has one output per bit in word new-i. If the strand is currently marked then the output gates of this circuit have value given by the word value new-word specified in the operation. If the circuit is currently unmarked then the output gates of this circuit have the value given by the variables *current-value*(new-i,j). Finally the variables *current-value*(new-i,j) are updated to refer to the outputs of this subcircuit. The circuit for the append-unmarked operation is similar.

If the tth operation is the unmark operation then the *marked* variable is updated to refer to the gate with fixed value 0. If the tth operation is a destroy-unmarked operation then a new **and**-gate g is added to the circuit which takes as input the gate named *marked* and the gate named *output*. Thus the value of gate g is 1 if and only if the input strand has not previously been destroyed and is not destroyed as a result of the current destroy-unmarked operation. Also the *output* variable is updated to refer to gate g. Finally if the tth operation is the erase operation then variable *marked* and the variables *present*(i), for all indices i not among the words initially on the strands, are updated to refer to the gate with fixed value 0. This completes the description of how the circuit may be constructed. □

Theorem 2 Restricted-Surface-Time($t(n)$) = Extended-Sat($t(n)$).

Proof: In proving both directions of this result we assume that both the algorithm and the extended-sat formula receive n-bit inputs, where the bits are organized into words.

Let F be an extended-sat formula with with n inputs. Suppose that F has t (unbounded fan-in) **or**-gates at level 1 and t' **and**-gates at level 2. The following restricted DNA algorithm computes exactly the set of satisfying inputs to F and uses exactly t' mark operations, t destroy-unmarked operations, and t unmark operations. The algorithm repeats the following procedure for each **or**-gate OR at level 1 of F: First for each **and**-gate that is a child of OR, perform one mark operation whose constraint specifies that the **and**-gate is set to 1. Once this is done for all **and**-gates that are children of OR, perform a destroy-unmarked operation followed by an unmark operation.

Conversely, let A be a restricted DNA algorithm with n inputs. Without loss of generality we can assume that the sequence of operations in A has the following structure. It is a sequence of blocks where each block is a sequence of mark operations, followed by a destroy-unmarked operation, followed by an unmark operation. It is not hard to verify that this structure is optimal for algorithms that do not compute the empty set. If A is organized into t blocks with a total of t' mark operations (and t destroy and t unmark operations at the end of each block) then it is straightforward to construct an extended-sat formula with a total of t **or**-gates at level 1 and t' **and**-gates at level 2 that accepts the same language as A. □

Theorem 3 Circuit-Size,Block($t(n), l(n)$) $\subseteq$ Surface-Time,Length($t(n), l(n)$).

Proof: The proof is a simple extension of the proof of Theorem 1. Let C be a blocked circuit with n inputs, t gates, and l blocks. The DNA algorithm that accepts the same language as C simply "simulates" each block in turn using the construction of Theorem 1 and performs an erase operation between blocks. As a result, $O(t)$ operations are performed in total and a maximum of $O(l)$ words are appended to any strand at any point in the computation. □

4 Conclusions

We have shown how to evaluate the output of a boolean circuit in parallel on many inputs using surface-based ma-

nipulation of DNA strands. The inputs are encoded using single-stranded DNA and the computation is done using the chemical processes of hybridization, ligation, polymerase extension, and exonuclease degradation.

There are many directions for further research. (i) Detailed designs for information storage on DNA strands need to be developed and the chemical processes used in each operation need to be refined and tested. (ii) Errors inherent in the operations need to be quantified and algorithmic methods for performing robust computation in the face of these errors need to be developed. (iii) It may be desirable to add new operations to the model, specifically operations that erase words from strands in a more flexible manner than that described here or operations that duplicate strands on the surface. (iv) It would be interesting to understand how the operations of our model may be useful in "computations" involving general DNA strands, that is, strands that are not restricted to encode binary information. For example, function computation in our model is akin to synthesis of sets of DNA. The company Affymetrix performs synthesis of complex sets of DNA using light-directed methods, in which bases are added one at a time to strands as a function of the location of a strand on the surface. In contrast, our model allows words to be added to a strand as a function of its content rather than its location. Finally, readout strategies need to be developed for both surface- and solution-based computation.

References

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

[2] Adleman, L. M. (1995). On Constructing a Molecular Computer, Manuscript, Computer Science Department, University of Southern California.

[3] Amos, M., Gibbons, A., and Hodgson, D. (1996). Error-resistant implementation of DNA computations, Proc. Second Annual Meeting on DNA-Based Computers, American Mathematical Society, to appear.

[4] Boneh, D., Dunworth, C., and Lipton, R. J. (1995). On the computational power of DNA, Princeton CS Technical Report CS-TR-489-95.

[5] Guo, Z., Guilfoyle, R. A., Thiel, A. J., Wang, R., and Smith, L. M. (1994). Direct fluorescence analysis of genetic polymorphisms by hybridization with oligonucleotide arrays on glass supports, Nucl. Acids Res., 22, 5456-5465.

[6] Jentsch, S. (1983). J. Bacteriol 156, 800-808.

[7] Lipton, R. J. (1995). DNA Solution of Hard Computational Problems, Science, 268, 542-545.

[8] Liu, Q., Guo, Z., Condon, A.E., Corn, R.M., Lagally, M.G., and Smith, L.M. (1996). A Surface-Based Approach to DNA Computation, Proc. Second Annual Meeting on DNA-Based Computers, American Mathematical Society, to appear.

[9] Mead, D. A., Pey, N. K., Herrnstadt, C., Marcil, R. A., and Smith, L. M. (1991). A Universal Method for the Direct Cloning of PCR Amplified Nucleic Acid, Biotechnology, 9(7), 657-663.

[10] Mir, K. U. (1996). A restricted genetic alphabet for DNA computing, Proc. Second Annual Meeting on DNA-Based Computers, American Mathematical Society, to appear.

[11] Pease, A. C., Solas, D., Sullivan, E. J., Cronin, M. T., Holmes, C. P., and Fodor, S. P. (1994). Light-generated oligonucleotide arrays for rapid DNA sequence analysis, Proc. Nat. Acad. Sci. USA, 91, 5022-5026.

[12] Smith, L. M. (1988). Automated synthesis and sequence analysis of biological macro-molecules, Analytical Chemistry 60, 381A-390A.