Fabricating Arrays of Strings

J. Richard Bradley
Steven S. Skiena*
Department of Computer Science
State University of New York
Stony Brook, NY 11794-4400
{bradley—skiena}@cs.sunysb.edu

Abstract

Interactive sequencing by hybridization (ISBH) is a new approach to DNA sequencing which uses interaction to dramatically reduce the number of oligonucleotides for de novo sequencing of large DNA fragments, while preserving the parallelism which is the primary advantage of conventional SBH. Through simulations, we have established that 64kb fragments can be sequenced using approximately ten rounds of standard-sized 64k oligonucleotide arrays. This represents a potentially significant advance in sequencing large DNA fragments.

The key to practical implementation of ISBH is the development of technologies for fabricating arrays of oligonucleotides which avoid substantial prefabrication costs which can only be amortized over large production runs. In this paper, we identify two such potential technologies, both of which are based on interesting combinatorial and algorithmic problems:

- *Designing fabrication schedules for non-regular arrays on the Southern Array Maker^{TM}* – The device constructs arrays by appending single characters to each cell along specific rows and columns of arrays. We resolve the complexity of constructing Southern arrays under three different models of construction, and present experimental results of a simulated annealing algorithm for constructing dense arrays. This algorithm is fundamental to our project of conducting the first laboratory experiments on ISBH, in collaboration with biochemists at Oxford University.
- *Identifying minimum mask sets for photolithographic arrays* – Standard photolithographic techniques for constructing SBH arrays require the preparation of expensive light masks customized to a particular design. For interactive SBH to be cost-effective under this model, masks must be reusable. We establish both upper and lower bounds on the function $M(n, t, r)$ – the fewest number of distinct masks sufficient to realize all 2^n masking patterns by superimposing up to t masks simultaneously in each of r rounds.

1 Introduction

Sequencing by hybridization (SBH) [2, 5, 6, 8, 13] is a new and promising approach to DNA sequencing which offers the potential of reduced cost and higher throughput over traditional gel-based approaches. Here, we consider issues inherent in a new approach to sequencing by hybridization which permits the sequencing of arbitrarily large fragments without the inherently exponential chip area of SBH, while retaining the massive parallelism which is the primary advantage of the technique.

The traditional sequencing by hybridization procedure attaches a set of single-stranded fragments to a substrate, forming a *sequencing chip*. A solution of single-stranded target DNA fragments is exposed to the chip. These fragments hybridize with complementary fragments on the chip, and the hybridized fragments can be identified using a fluorescent or phosphorescent dye. Each hybridization (or the lack thereof) determines whether the string represented by the fragment is or is not a substring of the target. The target DNA can now be sequenced based on the constraints of which strings are and are not substrings of the target. The surveys [3, 13] give an excellent overview of the current state of the art in sequencing by hybridization, both technologically and algorithmically.

Interactive sequencing by hybridization (ISBH) [10] uses *interaction* to reduce the required amount of work. After beginning with an experiment using a prefabricated sequencing chip (such as $C(8)$), we take the results from this experiment and use them to design a customized sequencing chip to resolve some of the remaining ambiguities. We repeat this process until the sequence is determined. The key issue is demonstrating that we can design good sequencing chips completing the job using few rounds of as few queries as possible. Table 1 demonstrates using less than ten rounds suffice to reduce the total number of queries by several order

*Supported by ONR award 400x116yip01 and NSF Grant CCR-9625669.

Fragment Length	Classical SBH Query Length	Size	Interactive SBH Rounds	Total Size
80	7	16,384	7	560
180	8	65,536	8	1,440
260	9	262,144	8	2,080
560	10	1,048,576	8	4,480
1300	11	4,194,304	9	11,700
2450	12	16,777,216	9	22,050

Table 1: Characteristic length of unambiguously deciphered DNA fragment as a function of the size for classical and interactive SBH.

	AA	AG	GA	GG
AA	AAAA	AAAG	AAGA	AAGG
AG	AGAA	AGAG	AGGA	AGGG
GA	GAAG	GAAG	GAGA	GAGG
GG	GGAA	GGAG	GGGA	GGGG

Figure 1: A prefix-suffix array of all purine 4-mers.

of magnitudes on reasonable fragment sizes. The potential savings increase very rapidly with the length of the fragment to be sequenced.

Having established the combinatorial advantages of ISBH via both analysis and simulation, the next step is to perform actual wet laboratory experiments, to study the robustness of the technique in the presence of errors. For ISBH to be practical, the critical issue is minimizing the cost of custom-designed sequencing arrays. In this paper, we identify two such potential technologies for doing this, both of which are based on interesting combinatorial and algorithmic problems.

- *Designing fabrication schedules for non-regular arrays on the Southern Array Maker*^{TM} – This device constructs arrays by appending single characters to each cell along specific rows and columns of arrays. Figure 1 illustrates a complete array of all purine 4-mers constructed as a Southern array. In this paper, we resolve the complexity of constructing Southern arrays under three different models of construction, and present experimental results of a simulated annealing algorithm for constructing dense arrays. This algorithm is fundamental to our goal of conducting the first laboratory experiments on ISBH in Fall 1996, in collaboration with biochemists at Oxford University.

  Biological applications of our results go well beyond ISBH, for the Southern Array Maker has already been used outside of Oxford for testing tandem repeats [20] and fingerprinting.

- *Identifying minimum mask sets for photolithographic arrays* – Standard photolithographic techniques for constructing SBH arrays require the preparation of expensive light masks customized to a particular design. For interactive SBH to be cost-effective under such a model, masks must be reusable. We establish both upper and lower bounds on the function $M(n,t,r)$ – the fewest number of distinct masks sufficient to realize all 2^n masking patterns by superimposing up to t masks simultaneously in each of r rounds.

In the sense of being biological techniques proposed by computer scientists, our work is philosophically akin to the recent work of Adleman [1] and Lipton [7] on biocomputing. As experimental procedures evolve for studying larger and hence more combinatorially sophisticated problems, we anticipate new opportunities for such collaboration between algorithmicists and biologists.

The outline of this paper is as follows. In Section 2, we present an overview of the state of the art in DNA sequencing by hybridization, and introduce interactive sequencing by hybridization in greater detail. The row/column-append array fabrication model of the Southern Array Maker is introduced in Section 3, with our theoretical results on constructing arrays in Section 4. Our simulated annealing solution and experimental results appear in Section 5. Our theory of realizing arrays using compound light masks is presented in Section 6. We conclude in Section 7 with open problems.

2 Sequencing by Hybridization

Sequencing by hybridization (SBH) is a new and promising approach to DNA sequencing which offers the potential of reduced cost and higher throughput over traditional gel-based approaches. In 1991, Strezoska, et.al. [19] accurately sequenced 100 base pairs of a known sequence using hybridization techniques, although the approach was proposed independently by several groups, including Bains and Smith [2], Drmanac and Crkvenjakov [5], Lysov, et.al [8], Macevicz [9], and Southern [16]. More recently, Crkvenjakov's and Drmanac's laboratories report sequencing a 340 base-pair fragment in a blind experiment [13].

In the classical sequencing chip $C(m)$, all 4^m single-stranded oligonucleotides of length m are attached to the surface of a substrate. For example, in $C(8)$ all $4^8 = 65,536$ octamers are used. Other, non-classical sequencing chip designs are possible. The economies of scale and parallelism implicit in performing thousands of hybridization experiments simultaneously are major advantages of prefabricated sequencing chips. The mass production of a prefabricated array can be used to amortize the high start-up costs of such a procedure, as is the case in the manufacture of VLSI chips. Indeed, Fodor, et. al. [6] use photolithography techniques typical of the semiconductor industry to fabricate the array $C(5)$ of 1024 peptides in only ten steps. Larger chips (such as $C(8)$) are currently being developed, and by the analogy with the semiconductor industry chip capacity can be expected to continue to grow exponentially for several years.

Interpreting the data is where the algorithmic aspect of sequencing by hybridization arises. Efficient algorithms exist for finding the shortest string consistent with the results of a classical sequencing chip experiment. In particular, Pevzner's algorithm for sequencing chip reconstruction [14] is based on finding Eulerian paths in a subgraph of the de Bruijn digraph [4].

2.1 Interactive Sequencing by Hybridization

Unfortunately, the size of classical sequencing chips grows exponentially, so only short DNA fragments may be sequenced on arrays of reasonable size. This motivates our study of interactive SBH.

Skiena and Sundaram [15] studied the complexity of *sequentially* reconstructing unknown strings from substring queries. Specifically, we show that $(\alpha - 1)n + \Theta(\alpha\sqrt{n})$ queries are sufficient to reconstruct an unknown string, where α is the alphabet size and n the length of the string, matching the information-theoretic lower bound for binary strings. Further, we show that $\sim \alpha n/4$ queries are necessary, which is within a factor of 4 of the upper bound for larger alphabets. However, achieving a high degree of parallelism is critical for this approach to lead to a practical method of DNA sequencing.

More recently, Margaritis and Skiena [10] studied the complexity of reconstructing unknown strings from rounds of parallel substring queries. We show a wide range of tradeoffs between the number of rounds of substring queries and the number of queries per round sufficient to determine an unknown string of length n on an alphabet of size α.

In [10], we proved an exponential lower bound on the capacity of any prefabricated sequencing chip capable of reconstructing n-strings, and give a new chip design whose capacity approaches this lower bound. Further, we give a strategy which uses (with probability $1-1/n^{\epsilon}$) $O(\alpha\epsilon \lg n)$ rounds of n queries per round. The success of this strategy has been validated by simulations – see [10] for details.

3 Southern Array Fabrication

A new and exciting approach for efficiently building oligonucleotide arrays uses the Southern Array Maker [17], manufactured by Beckman Instruments, to prepare discrete oligonucleotide sequences in parallel rows across a 6.6 cm $\times$ 6.6 cm polypropylene substrate [20]. This technology provides an ideal environment for testing the feasibility of interactive SBH in a laboratory, because with proper programming it will provide an inexpensive way to fabricate a wide variety of oligonucleotide arrays on demand.

We have begun a collaboration with David Coleman, in Ed Southern's laboratory at Oxford, which will lead to the first wet lab trials of interactive SBH. Several algorithmic issues need to be resolved to support this collaboration, which forms the focus of this paper. We anticipate that our algorithms and implementations will be useful in other laboratory applications of the Southern Array Maker outside of Oxford, such as testing for short tandem repeats [20] and fingerprinting. See [11, 18] for experimental results on hybridization error rates.

	A	C
G	AG	GC
T	TA	CT

	CG	GA
CT	CCTG	CTGA
CC	CGCC	CGCA

Figure 2: Two 2×2 oligonucleotide arrays. The left array is unrealizable using the Southern apparatus due to precedence constraints, while the right array is realizable.

First, we describe the Southern Array Maker^{TM} apparatus. The device consists of a specially designed 64-channel chemical delivery system, capable of applying different reagents along parallel rows or columns. Each reagent enables oligonucleotides to grow on the substrate by the addition of a single nucleotide of a specific type (either A, C, G, T or appropriate combinations like A/T, C/G, or A/C/G/T) to all molecules being grown along the track. In the case of combinations, one nucleotide is appended to each molecule, but because of the large number of molecules grown at the site this process occurs with probability roughly proportional to the nucleotide distribution in the mixture. Specific arrays are fabricated on this device by issuing a series of commands specifying either row or column vectors of oligonucleotides. Arbitrary combinations of nucleotides can be grown simultaneously along parallel tracks of the device.

3.1 Programming the Southern Array Maker

The Oxford device is currently being used for building $64 \times 64 = 4096$ oligonucleotide arrays, in the course of their research on the biochemistry of hybridization. To date, only regular arrays have been fabricated, such as the classical chip $C(6)$, $(ACGT)^3XX(ACGT)^3$ (where $X = A/C/G/T$), and $(ACGT)^3CG(ACGT)^3$, all of which can be built using simple sequences of commands.

Fabricating more complicated arrays requires solving a difficult combinatorial problem. We are given as input a set S of n strings (representing oligonucleotides) to fabricate on an $m \times m$ array (where $m = 64$ on the Southern apparatus), and we must produce a schedule of row and column commands to realize the set S.

Not all sets of $m \times m$ strings can be fabricated on the device. For example, $S = \{A, C, G, T\}$ cannot be fabricated on a 2×2 array, since no two strings contain common row or column characters. Figure 2(a) shows an unrealizable array for more subtle reasons – although the strings can be factored into common row and column subsequences, these subsequences cannot be interleaved without violating cyclic precedence constraints. On the other hand, Figure 2(b) shows an array which can be realized (with strings grown from left to right) by the following schedule: (1) C down left column, (2) C across top row, (3) T across top row, (4) G down left column, (5) C across bottom row, (6) G down right column, (7) C across bottom row, (8) A down right column.

There are three separate phases to array design: (1) assigning strings to positions in the array, (2) factoring out common row and column subsequences covering each string, and (3) scheduling row and column operations to realize the subsequences. All of these phases are non-trivial – indeed, we have shown that the problem of designing a schedule for a given set S is NP-complete under three separate restrictions, and hence theoretically intractable.

4 The Complexity of Array Construction

The general problem of constructing arrays on the Southern array maker may be defined as follows:
Instances: A set $S = \{s_1, s_2, ..., s_n\}$ of strings over an alphabet Σ, and two integers x and y.
Question: Does there exist a sequence A of row and column fill operations which realize S on an $x \times y$ array?

Unfortunately, the general problem of constructing arrays is hard, as shown in Section 4.1, even for strings of length at most three. The problem remains hard when restricted to the more restrictive class of prefix-suffix arrays, as discussed in Section 4.2, although positive results exist for fabricating complete arrays. However, we will present a simulated-annealing approach to prefix-suffix array construction in Section 5.

4.1 General Array Construction

Theorem 1 *The problem of Southern array construction is NP-complete, even for strings of length at most three.*

Proof: First, we show that the problem is in NP. At most $(x + y)l$ row/column operations are used in realizing any $x \times y$ array with strings of length l, so a schedule of operations can be tested in polynomial time.

To prove the hardness of string array construction we use a reduction from 3-satisfiability (3-SAT). Specifically, given an instance of 3-SAT which has m three-variable clauses, $C = \{c_1, c_2, ..., c_m\}$, drawn from a set of n variables, $V = \{v_1, v_2, ..., v_n\}$, we will construct an instance of the string array construction problem in which the set of strings S can be realized if and only if the instance of 3-SAT has a truth assignment to V which satisfies all clauses of C. We start by eliminating all clauses in C which include both a variable and its complement (e.g, $v_i, \neg v_i$). Such clauses are always satisfied and hence their removal does not affect the hardness of 3-SAT. In addition, we assume that for all $c_i, c_j \in C$ that $c_i \neq c_j$ if $i \neq j$; and that there is no single $v_k \in V$ such that v_k is a member of every clause in C. These assumptions, again, do not affect the hardness of 3-SAT.

The alphabet Σ contains a unique symbol ν_i for every variable $v_i \in V$, and a unique symbol $\overline{\nu}_i$ for the complement $\neg v_i$ of every variable $v_i \in V$. In addition, Σ contains the unique symbol "*". We use ϵ to denote the presence of the empty string in the array $G(x, y)$. We lexicographically order the elements of Σ such that

$$* < \nu_1 < \overline{\nu}_1 < \nu_2 < ... < \nu_n < \overline{\nu}_n$$

For each clause $c_i \in C$, we can compute the lexicographically ordered substrings of length 2 corresponding to the variables (or their negations) in c_i. Let $T = \{t_1, t_2, t_3, ..., t_l\}$ be the set of all such unique 2-substrings ("tuples"). Additionally, let $k = (n + l + 1)^5 \cdot (n + 1)^5$

The set S of strings to be realized on $G(x, y)$ consists of the following:

- For each clause $c_i \in C$ the string $\nu_a \nu_b \nu_c$ where $v_a, v_b, v_c \in c_i$; each of ν_a, ν_b, ν_c is either some ν_j or some $\overline{\nu}_j$; in the lexicographic ordering of Σ, $\nu_a < \nu_b < \nu_c$
- For each $v_i \in V$: (1) the string $\nu_i \overline{\nu}_i$; (2) the strings $*\nu_i$, $*\overline{\nu}_i$; and, (3) k copies each of the strings ν_i, $\overline{\nu}_i$
- For every $t_i \in T$: the string $*t_i$, and k copies of t_i
- The string "**"
- $2k$ copies of the string "*"
- k^2 copies of the string ϵ

Finally, $x = n + l + 1 + k$, and $y = n + 1 + k$.

Fix a satisfiable 3-SAT instance I. A solution to the string array construction problem instance I' corresponding to I appears in Table 2.

There are k entries for ϵ strings on both the rows and columns of the grid. The symbols ν_{i_h} denote symbols representing either a variable or its negation, while $\overline{\nu}_{i_h}$ is the symbol for the logical negation of the variable denoted by ν_{i_h}. Entries in the grid marked with $c_{j_{h'}}$ denote the realization of the string representing clause $c_{j_{h'}}$.

It can be seen that all strings in S are realized in the solution in Table 2.

The ordering of operations to realize this solution is derivable from the lexicographic ordering. All ϵ strings appear first in the sequence of operations. Next come "*" strings, followed by ν_i strings, and so on. For a given string array construction solution to be unrealizable due to an inability to successfully order fill operations it must be the case that some string s_i includes some two characters σ_a, σ_b such that the orderings $\sigma_a < \sigma_b$ and $\sigma_b < \sigma_a$ are forced by the general string layout. There are no required strings in S for which some two characters σ_a, σ_b occur in an order different from the lexicographic ordering defined above. Thus, issuing operations according to the lexicographic ordering on Σ ensures that all strings will be realizable.

To see that a solution to 3-SAT implies a solution to the string array construction problem, first let V_{sat} be a set of variables and variable negations in V which satisfies all clauses of C ($|V_{sat}| = n$). To derive the solution for 3-SAT instance I from the solution for the string array construction instance I' it is necessary only to set $V_{sat} = \{v_{i_1}, v_{i_2}, ..., v_{i_n}\}$, since only the variable strings $\{\nu_{i_1}, \nu_{i_2}, ..., \nu_{i_n}\}$ were used in the string array problem to realize clause strings. There are exactly n of these variable strings, and no variable string and the

	t_1	t_2	...	t_l	$\overline{\nu}_{i_1}$	$\overline{\nu}_{i_2}$	...	$\overline{\nu}_{i_n}$	$*$	ϵ	...	ϵ
ν_{i_1}			...	c_{j_1}	$\nu_{i_1}\overline{\nu}_{i_1}$		...		$*\nu_{i_1}$	ν_{i_1}	...	ν_{i_1}
ν_{i_2}	c_{j_2}	c_{j_3}	...			$\nu_{i_2}\overline{\nu}_{i_2}$	...		$*\nu_{i_2}$	ν_{i_2}	...	ν_{i_2}
ν_{i_3}			...				...		$*\nu_{i_3}$	ν_{i_3}	...	ν_{i_3}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ν_{i_n}			...	c_{j_m}			...	$\nu_{i_n}\overline{\nu}_{i_n}$	$*\nu_{i_n}$	ν_{i_n}	...	ν_{i_n}
$*$	$*t_1$	$*t_2$	...	$*t_l$	$*\overline{\nu}_{i_1}$	$*\overline{\nu}_{i_2}$	...	$*\overline{\nu}_{i_n}$	$**$	$*$	...	$*$
ϵ	t_1	t_2	...	t_l	$\overline{\nu}_{i_1}$	$\overline{\nu}_{i_2}$	...	$\overline{\nu}_{i_n}$	$*$	ϵ	...	ϵ
ϵ	t_1	t_2	...	t_l	$\overline{\nu}_{i_1}$	$\overline{\nu}_{i_2}$	...	$\overline{\nu}_{i_n}$	$*$	ϵ	...	ϵ
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
ϵ	t_1	t_2	...	t_l	$\overline{\nu}_{i_1}$	$\overline{\nu}_{i_2}$	...	$\overline{\nu}_{i_n}$	$*$	ϵ	...	ϵ

Table 2: String array construction solution to 3-SAT instance

symbol for that variable's logical negation both appear in $\{v_{i_1}, v_{i_2}, ..., v_{i_n}\}$. This implies that V_{sat} may be taken from the variables whose symbols were used to realize clause strings in the string array construction solution. Thus we have V_{sat} from our string array solution. Furthermore, this solution may be extracted in $O(n)$ time by reading off the strings in the operations used on the first n rows of the grid.

We must now show that, given a solution to instance I' of the string array construction problem we can produce, in polynomial time, a solution to instance I of 3-SAT.

We first note that any string s_i issued in an operation $a_i \in A$ must be a subsequence of some string in S, without loss of generality.

We define *mutually anti-social strings* to be those strings which, when used in row (column) fill operations, disallow the realization of any of the other mutually anti-social strings on that row (column) of G. The following strings in S are mutually anti-social with every string in S: ϵ; "**"; $\nu_h\overline{\nu}_h$, for any $1 \leq h \leq n$; the string representing c_i, for any $1 \leq i \leq m$; $*t_j$, for any $1 \leq j \leq l$.

Claim 1 *Generating the ϵ strings in S requires k column ϵ operations and k row ϵ operations.*

Proof: Using only row operations to realize the ϵ strings will require generation of the k copies of the strings $\{*, \nu_i, t_j\}$ by row operations as well. This is an artifact of the large size of k in relation to the values of n, m, and l, as well as the realization that since the strings in $\{*, \nu_i, t_j\}$ are mutually anti-social (to generate k copies of any one of these strings, a single operation must be dedicated to that string) they must be generated by distinct row/column fill operations. There are $(2n+2+l) \cdot k$ of these strings being generated, implying that we can use at most $(k+n+1) - (2n+2+l)$ rows for ϵ strings (due to the dimensions of G). This will allow the generation of $(k-n-l-1) \cdot (k+n+l+1) = (k-(n+l+1)) \cdot (k+(n+l+1))$ ϵ strings. Since $(n+l+1) > 0$, the number of ϵ strings generated is less than k^2.

Using only column operations to realize the ϵ strings will, by similar logic, allow the generation of only $(k + n + l + 1 - (2n+2+l)) \cdot (k+n+1) = (k-(n+1)) \cdot (k+(n+1)) < k^2$ ϵ strings.

Thus it follows that a conjunction of row and column ϵ operations must be used. Specifically, k row ϵ operations and k column ϵ operations are necessary. If fewer than k row (column) operations are used, more than k column (row) operations must be used to make up the deficit in generated ϵ strings. If, for some value of d, we can utilize $(k-d)$ row (column) ϵ operations and $(k+d)$ column (row) ϵ operations, we may exactly generate the k copies of the strings in $\{*, \nu_i, t_j\}$ by issuing row and column operations on the remaining (non-ϵ) area of G. Unfortunately, $(k-d) \cdot (k+d)$ is again less than k^2, implying that at least one additional row or column must be dedicated to generating ϵ strings. This indicates that some strings in $\{*, \nu_i, t_j\}$ will not be generated k times. ∎

Claim 2 *The strings in $\{*, \nu_i, t_j\}$ must be generated by $n+1$ distinct row operations and $n+l+1$ distinct column operations.*

Proof: Immediate from Claim 1. ∎

Claim 3 *Strings of the form $\nu_i\overline{\nu}_i$ must be generated by two distinct opposite fill operations, namely ν_i generated by a column (row) operation and $\overline{\nu}_i$ by a row (column) operation*

Proof: If the k copies of ν_i and $\overline{\nu}_i$ are both generated by row (column) operations, then an additional column (row) operation is necessary to generate the string $\nu_i\overline{\nu}_i$. This additional operation is antisocial with respect to the operations used to satisfy Claims 2 and 1, thus this new string must replace some operation used to generate strings in $\{*, \nu_i, t_j, \epsilon\}$ and hence some string in this set will not be generated a sufficient number of times. ∎

Claim 4 *All strings of the form t_j (tuple strings) are generated by distinct column operations.*

Proof: Claim 2 shows that tuple strings are generated by distinct fill operations. The enumeration of fixed operations at this point shows that $k+n$ row operations and $k+n$ column operations are dedicated to non-tuple strings, leaving potentially $l+1$ columns and a single row available to realize tuple strings. If a tuple string t_d is realized by a row operation, then Claim 3 asserts that all "*" strings are generated by columns, implying that the only string in S of the form $*t_j$ generated is $*t_d$. ∎

Claim 5 *The string "**" is generated by a row "*" operation and a column "*" operation.*

Proof: Due to the previous claims, the two fill operations which generate the $2k$ "*" strings may lie on (i) two rows, (ii) two columns, or (iii) a row and a column. Claim 3 implies that case (i) is impossible due to the fact that the $n+1$ non-ϵ rows contain n ν_i fill operations. Case (ii) does not allow the generation of all strings of the form $*t_j$ due to Claim 4. Since all available rows and columns are dedicated to operations involving $\{*, \nu_i, t_j, \epsilon\}$ the antisocial string "**" can not be used in a fill operation. ∎

At this point it is necessarily true that, apart from the ordering of rows and columns, that our solution to I' is of the form shown in Table 2. Most importantly, all clause strings must be generated by the intersection of variable symbols as row operations and tuple strings as column operations. Reading the list of variable symbols $\nu_{i_1}, \nu_{i_2}, ..., \nu i_n$ expressed as row operations produces a lists $V' = \{v_{i_1}, v_{i_2}, ..., v_{i_n}\}$ of variables or their negations, where each variable (or its negation) appears exactly once in V', and no variable and its negation are both in V'. V' constitutes a satisfying assignment (V_{sat} above) which solves instance I of 3-SAT. This solution was obtained in worst-case $O(k)$ time by examining the list of row operations and therefore the reduction is in polynomial time. ∎

4.2 Prefix-suffix array construction

We now consider a restricted version of the string array layout problem. We define the prefix-suffix string array layout problem as a version of the string array layout problem in which the only allowable fill operations consist of prefixes and suffixes of strings in S. Furthermore, all prefix operations must be issued on rows, while all suffix operations must be issued on columns, with all prefix operations issued prior to any suffix operations. Thus each generated string in S will be produced by a prefix operation followed by a suffix operation.

Table 1 shows the application of prefix-suffix operations to build a complete array of all 4-mers. All $2k$-mers can thus be realized with an $\alpha^k \times \alpha^k$ prefix-suffix array. All $2k+1$-mers can be realized with an $(\alpha^k + \alpha^{k+1}/2) \times (\alpha^k + \alpha^{k+1}/2)$ prefix-suffix array for even-sized alphabets. For all $2k+1$-mers on odd alphabets, an $(\alpha^k + \alpha^{k+1}) \times (\alpha^k + \alpha^k)$ prefix-suffix array suffices.

We show below that a restricted variant of this prefix-suffix string layout problem is NP-complete by a reduction from set cover. This hardness result will be used to show that even the general prefix-suffix string array layout problem is NP-complete on ternary alphabets.

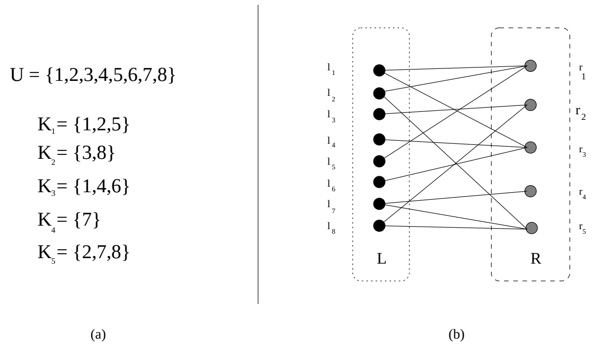

Figure 3: (a) an instance of set cover, and (b) its representation as a bipartite graph

4.2.1 Hardness of a restricted prefix-suffix problem

To show that the prefix-suffix variant of the string array layout problem is NP-complete, we first further restrict the problem. In addition to the above restrictions on usable operations, we restrict the available usable prefixes by taking as input to the prefix-suffix string array layout problem a set $P = \{p_1, p_2, ..., p_\rho\}$ of allowable prefixes. Only the prefixes in P may be used to generate the strings in S on $G(x, y)$.

Theorem 2 *The restricted prefix-suffix string array construction problem is NP-complete, even for ternary alphabets.*

Proof: Clearly the restricted prefix-suffix string array construction problem is in NP for the same reasons that the general string array construction problem was determined to be in NP in the proof of Theorem 1.

We use a reduction from set cover, which takes as input a *universal set* $U = \{u_1, u_2, ..., u_\delta\}$, κ unique subsets $K_1, K_2, ..., K_\kappa \subset U$, and an integer j. The decision problem is to determine whether there is a collection of no more than j of the K's whose set union contains all δ elements of U.

Any set cover instance can be modeled as a bipartite incidence graph $B = (R, L, E)$, where $V = R \cup L$. such that: An example is shown in Figure 3.

- Each element $u_i \in U$ will be represented by a vertex $l_i \in L$.
- Each set K_i will be represented by a vertex $r_i \in R$.
- An edge in B exists between two vertices r_i and l_j if and only if $u_j \in K_i$.
- A solution to the set cover problem corresponds to a size j subset of the vertices in R which covers every vertex in L.

We choose for our alphabet $\Sigma = \{0, 1\}$. For each element u_i in the universal set we create a string $s_i \in S =$ "$1^i01^{|R|}$" to be generated by prefix-suffix operations. For each element-subset incidence h between u_i and S_k we create a prefix $p_h \in P =$ "$1^k01^{|R|-i}$". For each subset S_i we define suf_i to be the string consisting of i "1" characters (i.e., "1^i"). Let SUF be the set of all such suffixes corresponding to subsets. Finally, let $x = j$ and $y = |L|$.

Each string in the array construction problem represents a universal set element in the original set cover problem. It may be seen that any string $s_i \in S$ may only be generated by those prefixes which correspond to set-element incidences. Since $x = j$, however, at most j suffixes could fit on the chip $G(x, y)$. Due to the construction of S only suffixes in SUF may be used in conjunction with the set of prefixes P. Therefore, the set of used suffixes corresponds to a set of no more than j vertices in L which together are adjacent to all vertices in R. Since this condition in B corresponds precisely to a solution to the original set cover problem, it follows that a solution to the restricted prefix-suffix array construction problem corresponds to a solution to the set cover problem.

Similarly, if there is a solution to our set cover instance, the bipartite graph representing this set cover instance will contain a set L' of at most j nodes in L which are together adjacent to all nodes in R. The corresponding array construction problem will have a solution which uses the suffixes corresponding to L' and the prefixes corresponding to edges adjacent to vertices in L' to generate the strings corresponding to the vertices in R. Thus a set cover instance is solvable if and only if the corresponding restricted prefix-suffix array construction problem is solvable.

Since the construction of B, the conversion of B into an array construction instance, and the extraction of a solution to set cover from array construction are clearly polynomial in time and space, it follows that the restricted prefix-suffix array construction problem is NP-complete. ∎

Theorem 3 *The general prefix-suffix string array construction problem is NP-complete.*

Proof: We again have a problem clearly in NP. Using the same reduction strategy used in the proof of Theorem 2, given a set cover instance, the bipartite graph B is constructed, and the set of strings S is taken from B in the same fashion. We amend the construction of Theorem 2 to include an additional alphabet character "*". Since the set P of prefixes cannot be used to restrict array construction, we add strings to S to force the use of the same prefix set P. Construct P using the same method as above. Let $x = j + (|E| \cdot |L|)^5$ and $y = |E| + 1$. For each prefix $p_i \in P$ add $x - j$ copies of "$p_i *$" to S. Finally, add $x - j$ copies of "**" to S.

If there is a solution to the set cover instance, then using the same prefixes and suffixes used in the proof of Theorem 2, along with a single "*" prefix, and $x - j$ "*" suffixes (as shown in Table 4.2.1) we can solve the corresponding prefix-suffix array construction problem.

Claim 6 *Any solution to the prefix-suffix array construction problem is of the form depicted in Table 4.2.1.*

Proof: The strategy used in this reduction is based upon logic similar to that used in the proof of Theorem 1. Specifically, the logic used in Claims 1- 5 above applies in this case, where failure to use P and "*" as prefixes as well as $x - j$ "*" suffixes will cause the omission of either "$p_i *$" strings, "**" strings, or strings in S due to the fact that at least one ϵ prefix must be used, and there will not be enough remaining space to generate all required strings. Once the prefixes and "*" are required, strings in S may only be generated by using the suffixes in SUF. ∎

If a solution to the array construction problem is found, its suffixes (other than "*") correspond to strings in SUF, therefore, by the logic of the proof of Theorem 2 we can derive a solution to the set cover problem. Similarly, we again have a polynomial time reduction, and thus the general prefix-suffix array construction problem is NP-complete. ∎

5 Experimental Results for Simulated Annealing

The formulation of the prefix-suffix array construction via bipartite graph covering used in the proof of Theorem 3 lends itself readily to attack via simulated annealing. We developed a program (hereafter `chip`) which uses simulated annealing to construct usable arrays given a set of strings as input. The LEDA data structures package [12] was used to ensure efficient manipulation of the graph and related data structures.

`chip` supports two modes of operation, "normal mode" and "reverse complement mode". Because it is possible to separate the two complimentary DNA strands and hybridize each separately against the array, for any desired probe we implement either it or its complementary string. By using this freedom, we may select whichever instantiation proves easier to fabricate. Note that since DNA sequences are directed strings, a probe and its reverse are not identical. Reverse complement mode supports greater array density at a cost of additional laboratory work.

Given a set S of strings, in either normal mode or reverse complement mode, `chip` constructs an initial array by building a complete array of all k-mers (strings of length k), where k is the average string length in S, as discussed in Section 4.2. As the array converges, the colored bipartite graph representing prefix-suffix relationships is constructed and kept in sync with the chip representation. Each prefix and suffix node in the graph is assigned a usefulness which measures the popularity of the prefix/suffix (the number of strings containing this prefix/suffix) and the length of the prefix/suffix (prefixes and suffixes which are nearly $k/2$ in length are favored over others).

Once the initial chip is constructed, the annealing code attempts to refine this chip by the application of the following random moves:

- *swap* – swap a prefix/suffix on the array with one that isn't.
- *add* – add a random prefix/suffix to the array.
- *delete* – delete a random prefix/suffix from the array.

	suf_{i_1}	suf_{i_2}	suf_{i_3}	...	suf_{i_j}	*	*	*	...	*
p_1	S_{h_1}			...		p_1*	p_1*	p_1*	...	p_1*
p_2		S_{h_2}		...		p_2*	p_2*	p_2*	...	p_2*
p_3				...	S_{h_3}	p_3*	p_3*	p_3*	...	p_3*
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
p_ρ	S_{h_β}			...		p_ρ*	p_ρ*	p_ρ*	...	p_ρ*
*						**	**	**	...	**

Table 3: Prefix-suffix array solution to set cover

	normal mode			reverse complement mode		
number of 7-mers	minimum	average	maximum	minimum	average	maximum
100	64	67	72	60	62.5	65
200	114	116.8	119	93	95.8	99
300	139	145.8	149	110	113.4	118
400	162	164.4	166	116	121.2	124
500	169	173.2	175	125	126.6	129
600	179	182	184	125	131.8	136
700	182	183.8	186	127	132.6	141
800	189	189.6	191	127	133	141
900	189	189.8	191	130	138.6	147
1000	190	191	192	130	142.8	148

Table 4: Maximum chip dimension statistics for sets of random 7-mers.

- *useful add* – add the prefix/suffix with the highest usefulness to the array.
- *useful delete* – delete the prefix/suffix with the lowest usefulness from the array.
- *string add* – randomly select a string not on the array, and add the most useful prefix and/or suffix which covers this string (additional preference is given to a prefix (suffix) whose corresponding suffix (prefix) is already on the array).

A standard annealing schedule is used, with an exponentially decreasing temperature (dependent upon the problem size) and a temperature-dependent Boltzmann criterion for accepting states which have higher costs than the known best. The cost function is defined as

$$cost = 2 \times max + min + \frac{(max - min)^2}{4} + 4(str_{total} - str_{in})$$

where max is the size of the maximum chip dimension, min is the size of the minimum chip dimension, $str_{total} = |S|$, and str_{in} is the number of strings in S currently on the chip.

Careful analysis of successful moves over the course of the annealing process suggested a second phase of annealing to speed convergence. Once the temperature reaches a predetermined cutoff point, the temperature schedule is changed to force the temperature to decrease more rapidly, and the probability distribution for choosing moves is altered to include only swap, add, and delete, with preference given to swap moves. This modification speeds late convergence, which has been typically slower than that in the early stages of the annealing process.

We conducted a series of test runs on sets of random 7-mers. For these trials we created five data sets of from 100 to 1000 7-mers on an alphabet of size 4. The simulated annealing algorithm was run three times on each set and the minimum value of the maximum array dimension taken as the result for that data set. Table 4 summarizes the minimum, maximum, and average values of these minimum values over all five datasets of a given size for both normal and reverse complement modes. Table 4 demonstrates that the expected size of the array grows quickly with the number of strings, before approaching a limit of the size of the complete array of 7-mers (192×192). It also shows the area savings resulting from reverse-complement mode. We note that higher densities will result from the more highly structured sets of strings generated by the ISBH algorithm of [10].

To demonstrate `chip`'s performance with real-world data, Figure 4 shows the convergence of a custom array to resequence the HIV-virus. Special-purpose arrays for diagnosis, such as the Affymetrix AIDS chip, represent the largest market for SBH arrays today. Splitting the sequence into its 5716 unique 7-mers, we used `chip` to construct a reverse complement array of the HIV 7-mers. Figure 4 shows snapshots of the state of the chip at four points during the annealing process (0, 500, 1000, and the final chip at 5750 iterations). Black pixels represent the first occurrence of an HIV 7-mer, while white pixels represent either duplicated HIV 7-mers, or strings not in the HIV input set. The final chip size here is 130×132, compressed from the initial

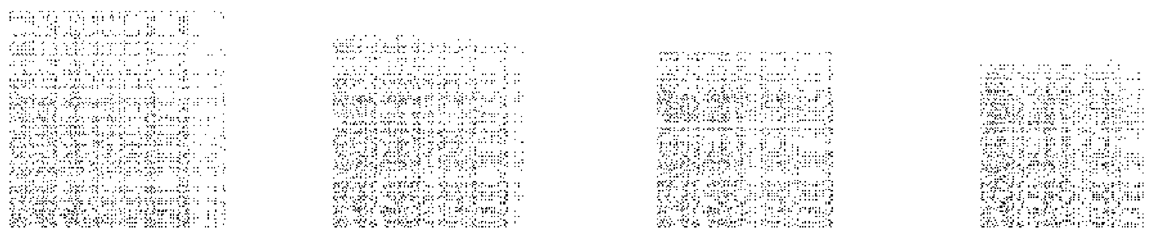

Figure 4: Compression of the HIV array by simulated annealing – after 0, 500, 1000, and 5750 iterations.

size of 192×192.

The run-time performance of `chip` proves sufficient to support laboratory use, taking on the order of minutes to construct arrays. All moves in the annealing process are implemented in $O(degree(u) + degree(v))$ time, where u and v are two graph nodes being manipulated. Issues of faster convergence, however, remain to be addressed. Establishing strong lower bounds on the size of chips, while further tuning the annealing mechanism should prove beneficial.

6 Photolithographic Technologies for ISBH

Although we propose to use the Southern Array Maker to implement interactive SBH for our initial laboratory trial, there are several other chip fabrication methods which appear promising for large-scale implementations of the method. The primary issues involve the costs for designing and fabricating new array designs. Certain proposed fabrication methods based on printing technologies, such as laser scanning and ink-jet spraying, imply a software chip design and place absolutely no premium on using a static chip design.

There appears to be no fundamental reason why the photolithography methods of [6] cannot be refined to construct customized sequencing chips with capacities of millions of oligonucleotides for ISBH. Affymetrix's photolithography technology relies on the use of light blocking masks which are placed between the chip and a light source, thereby allowing unmasked portions of the chip to be exposed to light. Subsequently, a specified base is chemically affixed to the areas of the chip which were light-exposed. In this fashion, arbitrary sequences of bases may be constructed by repeated application of masks and bases.

The primary cost of new designs with the Affymetrix technology involves the cost of etching the masks used to define light patterns on the substrate. Unfortunately, $\alpha \cdot k$ new masks are required to define every new array of k-mers on an alphabet of size α. This incremental cost could be eliminated if a small enough library of masks can be constructed to realize any desired light pattern. To realize any Boolean function on n array positions appears to require a library of 2^n masks, certainly prohibitive for reasonable values of n.

However, by superimposing masks we can obtain the equivalent of the mask formed by the union of the dark regions. Thus all 2^n patterns could be achieved using a stack of at most n masks from a library of n masks. In practice, however the thickness would be sharply limited by optical constraints to some small value t

Since the nucleotide growth reaction occurs only when exposed to light, we can partition a mask step into several (say up to s) disjoint steps (each of thickness up to t), provided each subsequent step covers all cells previously exposed.

We define the function $M(n,t,r)$ to be the number of distinct masks needed to expose any possible subset of the cells, given r rounds of thickness t. Our work on the masking SBH chip generation problem centers around bounding $M(n,t,r)$, and producing algorithmic mask generation methods which approach lower bounds placed on this function. Elementary bounds on $M(n,t,r)$ include:

- $M(n,n,1) = n$, realizable where mask M_i sets only the ith cell dark.
- $M(n,1,n) = n$, realizable where mask M_i sets only the ith cell light.
- $M(n,1,1) = 2^n$, realizable by constructing all masks.

Simple arguments suffice to prove more general bounds:

Theorem 4 $M(n,t,r) \geq 2^{n/rt}$.

Proof: At most m^t distinct masking patterns are possible using m distinct masks and a thickness of t. Thus at most $(m^t)^r \geq 2^n$ masking patterns can result from the composition of r rounds of masking. Taking the log of both sides gives the result. ∎

Theorem 5 $M(n,t,1) \leq t \cdot 2^{n/t}$.

Proof: Partition the n cells into t subsets of n/t cells each, $S_1, \ldots, S_t$. For each subset S_i, construct the $2^{n/t}$ masks representing each possible pattern of the elements. All 2^n possible patterns can be formed by selecting the correct mask for each disjoint subset. ∎

The problem of proving tighter bounds is left as an open problem.

7 Conclusions

We have studied the problem of constructing oligonucleotide array fabrication for the Southern Array Maker, both theoretically and experimentally. In particular, we have produced an array design program sufficient to support preliminary laboratory experiments of ISBH. Once the concept has been validated, other technologies for fabricating arrays will likely prove more appropriate in a production environment. Indeed, Oxford has started design for a pixel-addressable array maker, which would permit the fabrication of arbitrary arrays without loss of density, exactly what will be needed for production ISBH.

Other technologies have been proposed for ISBH, many of which suggest additional interesting problems:

- *Electrochemical deprotection technologies* — Another array fabrication technology (in use at Oxford) uses a small array of electrodes to deprotect the end of molecular chains electrochemically, thus permitting these chains to grow by an additional nucleotide. The effect is analogous to 'painting' a region of the substrate with one layer of a given nucleotide. Longer oligos can be constructed by painting additional layers, where these layers may partially overlap each other. Each region defined in the arrangement of paint spots defines a different oligo. Thus there is the following combinatorial problem: given a list of oligos to fabricate, what is the fabrication schedule which minimizes the total number of paint spots, or the movement of the sprayer?

- *Increasing the Density of Oligonucleotide Arrays* – Several ideas can be employed to increase the density of our oligonucleotide arrays. First, we have the freedom to employ mixtures of nucleotides, which gives us the possibility of placing tracks with mixed character positions, at a cost of potential ambiguity about which oligo actually hybridized. Such ambiguity can be eliminated by the reconstruction algorithm using knowledge of the previous and subsequent probes. Second, by controlling the temperature of the reaction, we can detect when partial hybridizations occur. How can we best design probes in such an environment?

- *Reuse of Oligonucleotide Arrays* – In the protocol we have outlined for interactive SBH, each synthesized sequencing chip is only used once. This appears wasteful since each array can be cleaned and reused many times. However, there is considerable potential to reuse chips and mask designs in sequencing mutant or homologous sequences, since the oligo content of these sequences is extremely similar.

Acknowledgements

We thank David Coleman, John Coleman, Jonathan Montagu, and Janos Pach for useful discussions.

References

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

[2] W. Bains and G. Smith. A novel method for nucleic acid sequence determination. *J. Theor. Biol.*, 135:303–307, 1988.

[3] A. Chetverin and F. Kramer. Oligonucleotide arrays: New concepts and possibilities. *Bio/Technology*, 12:1093–1099, 1994.

[4] N. G. de Bruijn. A combinatorial problem. *Koninklijke Nederlandse Akademie v. Wetenschappen*, 49:758–764, 1946.

[5] R. Dramanac and R. Crkvenjakov. DNA sequencing by hybridization. Yugoslav Patent Application 570, 1987.

[6] S. Fodor, J. Read, M. Pirrung, L. Stryer, A. Lu, and D. Solas. Light-directed, spatially addressable parallel chemical synthesis. *Science*, 251:767–773, 1991.

[7] R. J. Lipton. Speeding up computations via molecular biology. Princeton University, December 9, 1994.

[8] Y. Lysov, V. Florent'ev, A. Khorlin, K. Khrapko, V. Shik, and A. Mirzabekov. *Dokl. Acad. Sci. USSR*, 303:1508–, 1988.

[9] S. Macevicz. International Patent Application PS US89 04741, 1989.

[10] D. Margaritis and S. Skiena. Reconstructing strings from substrings in rounds. Proc. 36th IEEE Symp. Foundations of Computer Science (FOCS), October 22-25, 1995.

[11] U. Maskos and E. Southern. A study of oligonucleotide reassociation using large arrays of oligonucleotides sythesised on a glass support. *Nucleic Acids Research*, 21:4663–4669, 1993.

[12] K. Mehlhorn and S. Näher. LEDA, a platform for combinatorial and geometric computing. *Communications of the ACM*, 38:96–102, 1995.

[13] P. A. Pevzner and R. J. Lipshutz. Towards DNA sequencing chips. In *19th Int. Conf. Mathematical Foundations of Computer Science*, volume 841, pages 143–158, Lecture Notes in Computer Science, 1994.

[14] P.A. Pevzner. l-tuple DNA sequencing: Computer analysis. *J. Biomolecular Structure and Dynamics*, 7:63–73, 1989.

[15] S. Skiena and G. Sundaram. Reconstructing strings from substrings. *J. Computational Biology*, 2:333–353, 1995.

[16] E. Southern. United Kingdom Patent Application GB8810400, 1988.

[17] E. Southern. DNA chips: analysing sequence by hybridization to oligonucleotides on a large scale. *Trends in Genetics*, 12:110–114, 1996.

[18] E. Southern, U. Maskos, and J. Elder. Analyzing and comparing nucleic acid sequences by hybridization to arrays of oligonucleotides: evaluation using experimental models. *Genomics*, 13:1008–1017, 1992.

[19] Z. Strezoska, T. Paunesku, D. Radosavljevic, I. Labat, R. Drmanac, and R. Crkvenjakov. DNA sequencing by hybridization: 100 bases read by a non-gel-based method. *Proc. Nat. Acad. Science USA*, 88:10089–10093, 1991.

[20] M. Wehnert, R. Matson, J. Rampal, P. Coassin, and C. Caskey. A rapid scanning strip for tri- and dinucleotide short tandem repeats. *Nucleic Acids Research*, 22:1701–1704, 1994.