JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 11, Numbers 2–3, 2004
© Mary Ann Liebert, Inc.
Pp. 429–447

Scalable Heuristics for Design of DNA Probe Arrays

ANDREW B. KAHNG,[1] ION I. MĂNDOIU,[2] PAVEL A. PEVZNER,[3] SHERIEF REDA,[3] and ALEXANDER Z. ZELIKOVSKY[4]

ABSTRACT

Design of DNA arrays for very large-scale immobilized polymer synthesis (VLSIPS) (Fodor *et al.*, 1991) seeks to minimize effects of unintended illumination during mask exposure steps. Hannenhalli *et al.* (2002) formulate this requirement as the Border Minimization Problem and give an algorithm for placement of probes at array sites under the assumption that the array synthesis is synchronous; i.e., nucleotides are synthesized in a periodic sequence $(ACGT)^k$ and every probe grows by exactly one nucleotide with every group of four masks. Drawing on the analogy with VLSI placement, in this paper we describe and experimentally validate the engineering of several scalable, high-quality placement heuristics for both synchronous and asynchronous DNA array design. We give empirical results on both randomly generated and industry test cases confirming the scalability and improved solution quality enjoyed by our methods. In general, our techniques improve on state-of-the-art industrial results by over 4% and surpass academically published results by up to 35%. Finally, we give lower bounds that offer insights into the amount of available further improvements.

Key words: DNA array design, probe placement and embedding, optimization, algorithms, lower bounds.

1. INTRODUCTION

DNA PROBE ARRAYS, OR DNA CHIPS, have emerged as a core genomic technology that enables cost-effective gene expression monitoring, mutation detection, single nucleotide polymorphism analysis, and other genomic analyses (see Lipshutz *et al.* [1999] for a survey). DNA chips are manufactured through a highly scalable process, called *very large-scale immobilized polymer synthesis (VLSIPS)*, which combines photolithographic technologies adapted from the semiconductor industry with combinatorial chemistry. Commercially available DNA chips contain more than half a million probes and are expected to exceed one hundred million probes in the next generation (Lipshutz *et al.*, 1999). The design of DNA arrays raises

Preliminary versions of the results in this paper have appeared in Kahng *et al.* (2002, 2003).

[1]Departments of Computer Science and Engineering and of Electrical and Computer Engineering, UC San Diego, La Jolla, CA 92093.

[2]Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269-3155. Work performed in part while the author was with the Department of Electrical and Computer Engineering, UC San Diego.

[3]Department of Computer Science and Engineering, UC San Diego, La Jolla, CA 92093.

[4]Department of Computer Science, Georgia State University, Atlanta, GA 30303.

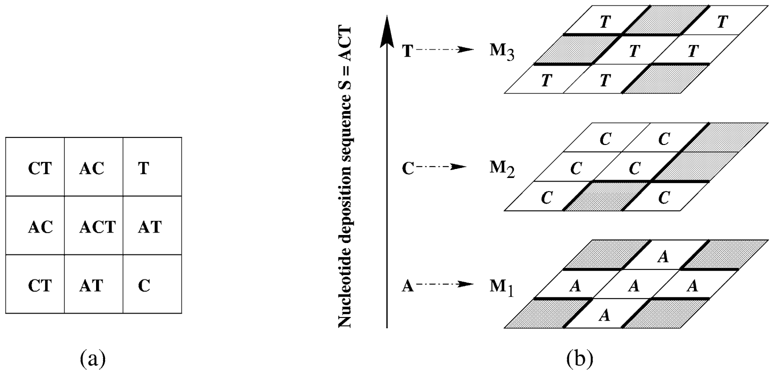

FIG. 1. (**a**) Two-dimensional probe placement. (**b**) Three-dimensional probe embedding: the nucleotide deposition sequence $S = (ACT)$ corresponds to the sequence of three masks M_1, M_2, and M_3. In each mask, the masked sites are shaded, and the borders between transparent and masked sites are thickened.

a number of challenging combinatorial problems, such as probe selection (Li and Stormo, 2000; Rahmann, 2002), deposition sequence design (Kasif *et al.*, 2002; Tolonen *et al.*, 2002; Rahmann, 2003), probe placement (Hannenhalli *et al.*, 2002), manufacturing quality control (Alon *et al.*, 2001; Hubbell and Pevzner, 1999; Sengupta and Tompa, 2002), etc. In this paper, we study the Border Minimization Problem that was recently introduced by Hannenhalli *et al.* (2002).

As described by Fodor *et al.* (1991), during VLSIPS the *sites* of a DNA probe array are selectively exposed to light in order to activate oligonucleotides for further synthesis. The selective exposure is achieved by a sequence $M_1, M_2, \ldots, M_K$ of *masks*, with each mask M_i inducing deposition of a particular nucleotide $s_i \in \{A, C, T, G\}$) at its clear array sites. The *nucleotide deposition sequence* $S = s_1 s_2 \ldots s_K$ corresponding to the sequence of masks is therefore a supersequence of all probe sequences in the array. Typically, S is assumed to be periodic, e.g., $S = (ACGT)^k$, where $(ACGT)$ is a *period* and k is the (uniform) length of all probes in the array.

Optical effects (diffraction, reflections, etc.) can cause unwanted illumination at masked sites that are adjacent to the sites intentionally exposed to light, i.e., at the *border* sites of clear regions in the mask. This results in synthesis of unforeseen sequences in masked sites and compromises interpretation of experimental data. To reduce such uncertainty, one can exploit the freedom available in how the probes are assigned to array sites. The *Border Minimization Problem (BMP)* (Hannenhalli *et al.*, 2002) seeks a placement of probes that minimizes the sum of border lengths in all masks.

Observe that, in general, a given probe can be *embedded* within the nucleotide deposition sequence S in several different ways.[1] We may view the array design as a *three-dimensional placement problem* (see Fig. 1): two dimensions represent the sites of the array, and the third dimension represents the deposition sequence S. Each layer in the third dimension corresponds to a mask inducing deposition of a particular nucleotide (A, C, G, or T), and each column within this three-dimensional placement representation corresponds to a probe embedded into the deposition sequence S. The border length for a given mask is computed as the number of *conflicts*, i.e., pairs of adjacent transparent and masked sites in the mask. Given two adjacent embedded probes p and p', the *conflict distance* $d(p, p')$ is the number of conflicts between the corresponding columns. The total border length of a three-dimensional placement is the sum of conflict distances between adjacent probes.

We distinguish two types of DNA array synthesis. In *synchronous* synthesis, the i^{th} period ($ACGT$) of the periodic nucleotide deposition sequence S synthesizes a single nucleotide (the i^{th}) in each probe. This corresponds to a unique (and trivially computed) embedding of each probe p in the sequence S; see Fig. 2(a–b). On the other hand, *asynchronous* array synthesis permits arbitrary embeddings, as shown in Fig. 2(c–d).

[1]An embedded probe q is defined as a sequence of length $K = |S|$ over the alphabet $\{A, C, G, T, b\}$ such that the j^{th} letter of q is either b (for blank) or s_j, the j^{th} letter of the nucleotide deposition sequence S.

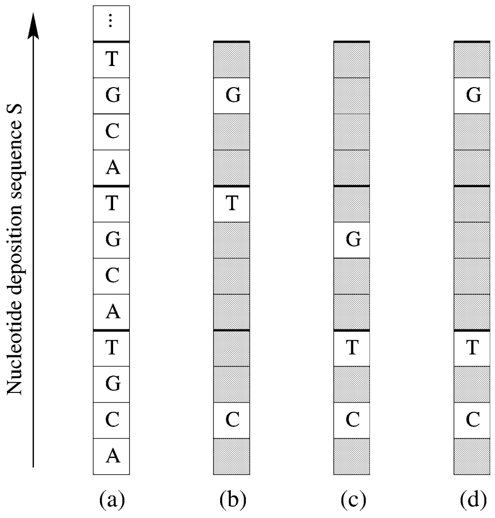

FIG. 2. (**a**) Periodic nucleotide deposition sequence S. (**b**) Synchronous embedding of probe *CTG* into S; the shaded sites denote the masked sites in the corresponding masks. (**c–d**) Two different asynchronous embeddings of the same probe.

Our contributions are as follows:

1. **Improved assignment of probes to sites for synchronous array synthesis.** Previous work on DNA array synthesis has considered only the synchronous context, when the conflict distance between two probes is $d(p, p') = 2h(p, p')$, with $h(p, p')$ denoting the Hamming distance between p and p' (i.e., the number of positions in which p and p' differ). As recounted by Hannenhalli *et al.* (2002), the first array design at Affymetrix used a traveling salesman problem (TSP) heuristic to arrange all probes in a tour that heuristically minimized Hamming distance between neighboring probes in the tour. The tour was then *threaded* into the two-dimensional array of sites. Hannenhalli *et al.* (2002) enhanced this threading approach to achieve up to 20% border length reduction for large chips. In Section 3, we suggest *epitaxial* placement heuristics that start by placing a random probe in the center or corner of the array and then continue to insert probes in sites adjacent to already-placed probes, so as to greedily minimize the number of induced conflicts.[2] We also demonstrate the value of simple ordering-based methods for initial placement and propose the use of a scalable sliding-window technique having antecedents in large-scale integrated circuit placement (Doll *et al.*, 1994; Huang and Kahng, 1997; Preas and Lorenzetti, 1988; Shahookar and Mazumder, 1991), as well as a local improvement operator based on optimal reassignment of an independent set of probes. A recurring motif is the analogy between silicon chip design and DNA chip design, pointing to the value of technology transfer between the 40-year old VLSI CAD field and the newer realm of probe array design. Experimental results confirm the linear run-time scaling and improved solution quality compared to previous methods.
2. **Dynamic programming algorithms for optimal embedding of single probes and iterative algorithms for full chip in-place optimization of probe embedding.** Note that in the asynchronous context, the conflict distance between two adjacent probes depends on their embedding. Section 4 proposes dynamic programming algorithms that optimally embed a given probe with respect to fixed embeddings of the probe's neighbors. This dynamic programming algorithm is used as the key subroutine in methods for iterative in-place optimization of probe embeddings. We propose and compare three such methods, referred to as *Batched Greedy*, *Chessboard*, and *Sequential Alignment* algorithms.
3. **Lower bounds for synchronous and asynchronous array design problems.** In Section 2 we give a priori lower bounds on the total border length of the optimum synchronous solution based on Hamming distance and of the optimum asynchronous solution based on the length of the longest common sub-

[2]A similar heuristic has been recently explored by Abdueva and Skvortsov (personal communication).

sequence (LCS). These give an estimate of the potential for future improvements in probe placement algorithms. In Section 4, a tighter LCS-distance-based lower bound is obtained for the in-place probe embedding problem, yielding bounds on possible improvement from exploiting this degree of freedom alone.

4. **Engineering of a scalable, high-quality flow for asynchronous DNA array design.** In Section 5, we give details on how to handle practical extensions and constraints, such as distance- and position-dependent border conflict weights and the presence of polymorphic probes (SNPs) in the instance. Throughout the paper, we describe and experimentally validate numerous algorithmic and implementation choices. Finally, in Section 6 we give experimental results on both randomly generated and industry test cases showing that our approaches are highly scalable and give placements of higher quality compared to previous methods.

2. ARRAY DESIGN PROBLEM FORMULATIONS AND LOWER BOUNDS

In this section, we give graph-theoretical formulations for the synchronous and asynchronous variants of the array design problem. For both variants, we give lower bounds on the cost of optimum solutions.

Following the development of Hannenhalli *et al.* (2002), let $G_1(V_1, E_1, w_1)$ and $G_2(V_2, E_2, w_2)$ be two edge-weighted graphs with weight functions w_1 and w_2. (In the following, any edge not explicitly defined is assumed to be present in the graph with weight zero.) A bijective function $\phi : V_2 \to V_1$ is called a *placement* of G_2 on G_1. The cost of the placement is defined as

$$cost(\phi) = \sum_{x,y \in V_2} w_2(x, y) w_1(\phi(x), \phi(y)).$$

The *optimal placement problem* is to find a minimum cost placement of G_2 on G_1.

The border minimization problem for synchronous array design can be cast as an optimal placement problem. In this case, we let $G2$ be a *two-dimensional grid graph* corresponding to the arrangement of sites in the DNA array; i.e., $V(G2)$ has $N \times N$ vertices corresponding to array sites, and $E(G2)$ has edge weights of 1 for every vertex pair corresponding to adjacent sites and edge weights of 0 otherwise. Also, let H be the *Hamming graph* defined by the set of probes, i.e., the complete graph with probes as vertices and each edge weight equal to twice the Hamming distance between corresponding probes. The border minimization problem for synchronous array design can then be formulated as follows:

Synchronous Array Design Problem (SADP). Find a minimum-cost placement of the Hamming graph H on the two-dimensional grid graph $G2$.

Let L be the directed graph over the set of probes obtained by including arcs from each probe to the four closest probes with respect to Hamming distance and then deleting the heaviest $4N$ arcs. Since the total weight of L cannot exceed the conflict cost of any valid placement of H on the grid graph $G2$, we obtain the following:

Theorem 1. *The total arc weight of L is a lower bound on the cost of the optimum SADP solution.*

For asynchronous array design, formalizing BMP is more involved. Conceptually, asynchronous design consists of two steps: (i) embedding each probe p into the nucleotide deposition sequence S, and (ii) placing the embedded probes into the $N \times N$ array of sites. Let H' be the complete graph with vertices corresponding to the embedded probes and with edge weights equal to the Hamming distance between them.[3] The border minimization problem for asynchronous array design can then be formulated as follows:

[3]Recall that embedded probes are viewed as sequences of length $K = |S|$ over the alphabet $\{A, C, G, T, b\}$ such that the j^{th} letter is either b or s_j. Thus, conflicts between two adjacent embedded probes occur only on positions where a nucleotide in one probe corresponds to a blank in the other.

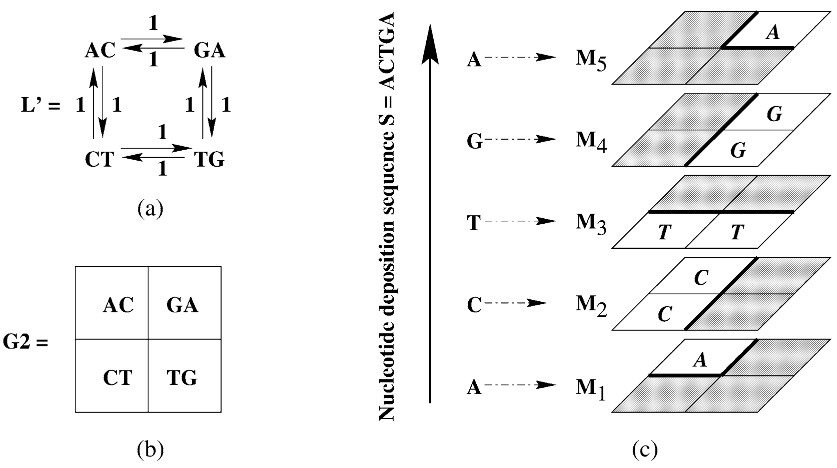

FIG. 3. (**a**) Lower-bound digraph L' for the probes *AC*, *GA*, *CT*, and *TG*. The arc weight of L' is 8. (**b**) Optimum two-dimensional placement of the probes. (**c**) Optimum embedding of the probes into the nucleotide deposition supersequence $S = ACTGA$. The optimum embedding has 10 conflicts, exceeding the lower bound by 2.

Asynchronous Array Design Problem (AADP). Find embeddings into the nucleotide deposition sequence S for all given probes and a placement of the corresponding graph H' on the two-dimensional grid graph $G2$ such that the cost of the placement is minimized.

In order to obtain nontrivial lower bounds on the cost of the optimum AADP solution, it is necessary to establish a lower bound on the conflict distance between two probes independent of their embedding into S. We get such a lower bound by observing that the number of nucleotides (mask steps) common to two embedded probes cannot exceed the length of the *longest common subsequence* (LCS) of the two probes. Define the *LCS distance* between probes p and p' by $lcsd(p, p') = k - |LCS(p, p')|$, where $k = |p| = |p'|$, and let L' be the directed graph over the set of probes obtained by including arcs from each probe to the four closest probes with respect to LCS distance and then deleting the heaviest $4N$ arcs. Similar to Theorem 1 we get:

Theorem 2. *The total arc weight of* L' *is a lower bound on the cost of the optimum AADP solution.*

Example. We remark that the weight of L' may be smaller than the optimum cost, since the embeddings needed to achieve LCS distance between pairs of adjacent probes may not be compatible with each other. Figure 3 gives one such example consisting of four dinucleotide probes, *AC*, *GA*, *CT*, and *TG*, which must be placed on a 2×2 grid. In this case, the lower bound on the number of conflicts is 8 while the optimum number of conflicts is 10.

3. SCALABLE ALGORITHMS FOR SADP

In this section, we describe the engineering of near-linear-time, yet high-quality synchronous probe placement heuristics. A recurring motif in our discussion is the value of technology transfer between the 40-year VLSI design literature and the newer field of DNA chip design. We point out analogies that inspire useful heuristics, as well as distinctions that hamper direct transfers. Two key design goals are (i) to enable scaling to 100 million or more placeable objects in the near future (say, within the current or next generation of workstations) and (ii) to enable easy parallelism (implying near-linear speedup on workstation clusters) wherever possible. We first describe an epitaxial growth algorithm inspired from the VLSI design literature and then describe a scalable version of it which we call row-epitaxial. Finally, we give a highly scalable heuristic based on optimally re-placing an independent set of probes; again using placement improvement ideas from VLSI design.

```
Input: Set $P$ of $N^2$ probes, scaling coefficients $k_i, i = 1,\ldots,3$
Output: Assignment of the probes to the sites of an $N \times N$ grid

1. Mark all grid sites as empty
2. Assign a randomly chosen probe to the center site and mark this site as full
3. While there are empty sites, do
       If there exists an empty site $c$ with all 4 neighbors full, then
           Find probe $p(c) \in P$ with minimum sum of Hamming distances to the neighboring probes
           Assign probe $p(c)$ to site $c$ and mark $c$ as full
       Else
           For each empty site $c$ with $i > 0$ adjacent full sites, find probe $p(c) \in P$ with minimum sum $S$ of Hamming
               distances to the probes in full neighbors, and let $norm\_cost(c) = k_i S/i$.
           Let $c^*$ be the site with minimum $norm\_cost$
           Assign probe $p(c^*)$ to site $c^*$ and mark $c^*$ as full
```

FIG. 4. The Epitaxial Algorithm.

3.1. Epitaxial growth SADP algorithms

In this section, we describe the so-called *epitaxial placement* approach to SADP and discuss some efficient implementation details. Epitaxial, or seeded crystal growth, placement is a technique that has been well explored in the VLSI circuit placement literature (Preas and Lorenzetti, 1988; Shahookar and Mazumder, 1991). The technique essentially grows a two-dimensional placement around a single starting seed.

The algorithm of Hannenhalli *et al.* (2002), which finds a TSP tour and then threads it into the array, optimizes directly only half of the pairs of adjacent probes in the array (those corresponding to tour edges). Intuitively, the epitaxial algorithm (see Fig. 4) attempts to make full use of the available information during placement. The algorithm places a random probe at the center of the array and then iteratively places probes in sites adjacent to already-placed probes so as to greedily minimize the average number of conflicts induced between *all* newly created pairs of neighbors. We have found that sites with more filled neighbors should have higher priority to be filled. In particular, we give highest priority to sites with four known neighbors. In the remaining cases, we apply scaling coefficients to prioritize candidate probe-site pairs. In our implementation, if a probe is to be placed at a site with $i < 4$ placed neighbors, then the average number of conflicts caused by this placement is multiplied by a coefficient $0 < k_i \leq 1$. Based on our experiments, we set $k_1 = 1$, $k_2 = 0.8$, and $k_3 = 0.6$. In our implementation, we avoid repeated distance computations by keeping with each border site a list of probes sorted by normalized cost. For each site, this list is computed at most four (and on the average two) times, i.e., when one of the neighboring sites is being filled while the site is still empty.

While the epitaxial algorithm achieves better results compared to other two-dimensional placements, it becomes impractical for DNA chips with dimensions of 300×300 or more. We note the following.

Observation 1. *Any placement method can be trivially scaled by partitioning the set of probes and the probe array into K subsets (chunks) each, then solving K independent placement problems. While runtime remains linear in the number of probes, two types of losses are incurred: (i) those from lack of freedom of a probe to move anywhere other than its subset's assigned chunk of array sites and (ii) those from lack of optimization on borders between chunks.*

Based on Observation 1, we have implemented trivial scaling for the epitaxial algorithm using chunk sizes up to 50x50. However, the results are dominated by those obtained using the following scalable variant of the epitaxial algorithm, which we call the *row-epitaxial* algorithm. There are three main distinguishing features of the row-epitaxial variant:

(1) It reshuffles an existing preoptimized placement rather than starting with an empty placement.
(2) The sites are filled with crystallized probes in a predefined order, namely, row by row and within a row from left to right.
(3) The probe filling each site is chosen as the best candidate not among all remaining ones, but among a bounded number of them (among the not yet "crystallized" probes within the next k_0 rows in our implementation, where k_0 is a parameter of the algorithm).

Feature (1) is critical for compensating the loss in solution quality due to the reduced search space imposed by (2) and (3). Since the initial placement must be very fast to compute, we cannot afford using any two-dimensional placement based on computing all pairwise distances between probes (such as TSP-based placement in Hannenhalli *et al.* [2002]). Possible initial placement algorithms can be based on space-filling curve (e.g., Gray code) ordering (Bartholdi and Platzman, 1982); indeed such orderings have had success in the VLSI context (Alpert and Kahng, 1994). We found that excellent initial placements are obtained by simply ordering the probes lexicographically (this can be done in linear time by radix sort) and then threading them as in Hannenhalli *et al.* (2002). Features (2) and (3) speed up the algorithm significantly, with the number k_0 of look-ahead rows allowing a fine tradeoff between solution quality and runtime.

3.2. Highly scalable algorithms for synchronous placement improvement

In the early VLSI placement literature, iterative placement improvement methods relied on weak neighborhood operators such as pair-swap, leveraged by metaheuristics such as simulated annealing. More recently, strong neighborhood operators have been proposed which improve larger portions of the placement. For example, the DOMINO approach (Doll *et al.*, 1994) iteratively determines an optimal reassignment of all objects within a given window of the placement. The end-case placer of Caldwell *et al.* (1999) uses branch and bound to optimally reorder small subrows of a row-based placement. Extending such improvement operators to full-chip scale, such that placeable objects can eventually migrate to good locations within practical runtimes, is typically achieved by shifting a fixed-size sliding window (Doll *et al.*, 1994) around the placement; cf. cycling and overlapping (Huang and Kahng, 1997), row-ironing (Caldwell *et al.*, 1999), etc.

For DNA arrays, an initial placement (and embedding) of probes in array sites may be improved by changing the placement and/or the embedding of individual probes. Guided by the VLSI experience and the intuition that randomly chosen pairs of optimally embedded probes are extremely unlikely to be swappable with reduction in border cost, we focus on strong operators. We make the following observation:

Observation 2. *Simultaneous probe re-placement of an entire window is not practical even for very small windows, but simultaneous probe re-placement of an independent set within a window is practical.*

Observation 2 rules out direct analogs of the DOMINO (Doll *et al.*, 1994) and end-case placer (Caldwell *et al.*, 1999) VLSI placement approaches. Instead, we propose the following novel method of improving the placement solution within a small region of the array. While improvements are still possible, we choose a set of mutually nonadjacent (independent) array cells, then re-place the probes in these cells according to a minimum-cost assignment, where the cost of assigning probe p to cell c is given by the sum of Hamming distances to the four neighbors. For a set of t independent cells, computing the minimum cost assignment requires $O(t^3)$ time. Full-chip application with practical runtime is achieved by iteratively choosing the independent set from a sliding window that is moved around the array; this approach is a reminiscence of early work on electronic circuit placement by (Akers, 1981; Steinberg, 1961).

We have carefully studied a number of methods for choosing the independent set within a window: (i) random maximal independent set, (ii) chessboard-based independent set (white squares or black squares), (iii) best result from among K different maximal independent sets, etc. We have found that using a single random maximal independent set ($K = 1$) yields the best tradeoff between solution quality and runtime. Therefore, we have implemented the above sliding window method as follows. (1) We first radix-sort all

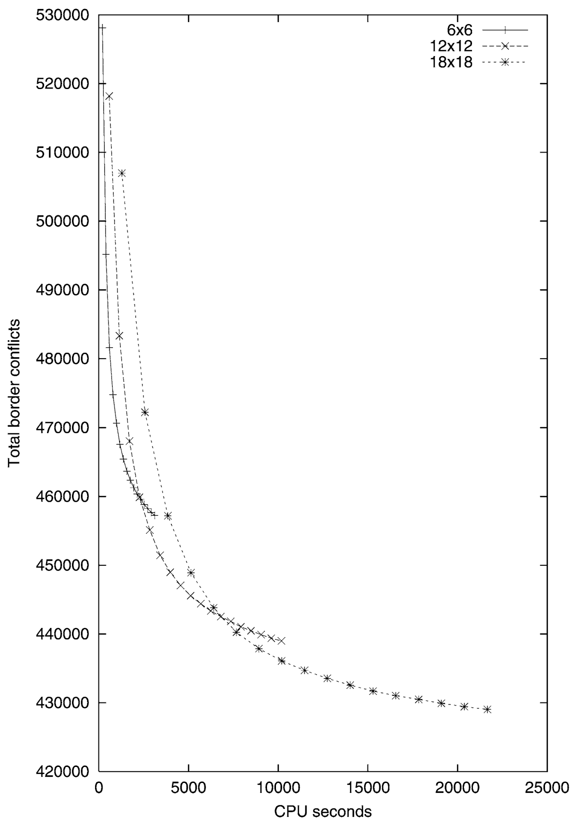

FIG. 5. Solution quality versus runtime for Synchronous Sliding-Window Matching with varying window size; array size = 100 × 100.

probes lexicographically and then perform 1-threading as in Hannenhalli *et al.* (2002). (2) Then, for each sliding $W_0 \times W_0$ window, we choose one random maximal independent set of sites and determine the cost of (asynchronous) reassignment of each associated probe to each site, then reassign probes according to the minimum weight perfect matching in the resulting weighted bipartite graph. (3) The window slides in rows, beginning in the top-left corner of the array; at each step, it slides horizontally to the right as far as possible while maintaining a prescribed amount of *window overlap*. After the right side of the array is reached, the window returns to the left end of the next row while maintaining the prescribed overlap with the preceding row. When the bottom side of the array is reached, the window returns to the top-left corner. (4) The window-sliding continues until an entire pass through the array results in less than 0.1% reduction of border cost.[4]

Our experiments (Kahng *et al.*, 2003) have shown that an overlap equal to half the window size gives best results; we use this setting for all results reported in this paper. Figure 5 illustrates the heuristic tuning with respect to varying window sizes. We observe that larger window sizes lose out to smaller window sizes when CPU time is very limited. Unless otherwise noted, experimental results below are obtained with window size = 6 (i.e., 6×6) and window overlap = 3 (for these values, the typical size of the random maximal independent set is around 13). Other experiments have shown that more effort in each window (and fewer cycles) loses out to less effort in each window (and more cycles); i.e., being greedier within a single window thwarts overall solution quality. Specifically, using multiple iterations of independent-set matching within a given window, or choosing the best of several attempted independent-set matchings, worsens our results. Last, we emphasize that the Sliding-Window Matching algorithm is easily parallelizable after the

[4]Faster variants can restrict the number of such passes to a small constant, e.g., 5. For arrays with up to half a million probes, our implementation makes around 20 passes.

TABLE 1. TOTAL BORDER COST, GAP FROM THE LOWER BOUND GIVEN BY THEOREM 1, AND CPU SECONDS (AVERAGES OVER 10 RANDOM INSTANCES) FOR THE TSP HEURISTIC OF HANNENHALLI *et al.* (2002) (TSP+1THR), THE ROW-EPITAXIAL (ROW-EPITAXIAL), AND THE SLIDING-WINDOW MATCHING (SWM) HEURISTIC[a]

Chip	*Lower bound*	*TSP+1Thr*			*Row-epitaxial*			*SWM*		
size	*Cost*	*Cost*	*Gap (%)*	*CPU*	*Cost*	*Gap (%)*	*CPU*	*Cost*	*Gap (%)*	*CPU*
100	410019	554849	35.3	113	502314	22.5	108	605497	47.7	2
200	1512014	2140903	41.6	1901	1913796	26.6	1151	2360540	56.1	8
300	3233861	4667882	44.3	12028	4184018	29.4	3671	5192839	60.6	19
500	8459958	12702474	50.1	109648	11182346	32.2	10630	13748334	62.5	50

[a]We use an upper bound of 20000 on the number of candidate probes in Row-Epitaxial (i.e., we use $k_0 = 20000/\textit{chipsize}$ look-ahead rows), and 6×6 windows with overlap 3 for SWM.

initial (linear-time) sorting and 1-threading step; the previous methods of Hannenhalli *et al.* (2002) do not have this property.

3.3. Empirical evaluation of synchronous placement algorithms

Table 1 compares the cost (number of conflicts) and runtime for the new synchronous placement algorithms and the TSP heuristic of Hannenhalli *et al.* (2002). Experiments were performed on arrays of 25-mer probes chosen uniformly at random. TSP+1-Threading runtimes are for an SGI Origin 2000 with 16 195-MHz MIPS R10000 processors (only one of which is actually used by the sequential implementations included in our comparison) and 4 GB of internal memory, running under IRIX 6.4 IP27. Row-epitaxial and SWM runtimes are for a dual-processor 1.4-GHz Intel Xeon server with 512 MB RAM. For comparison, we include the synchronous placement lower bound given by Theorem 1; the columns labeled "%Gap" show by how much the respective heuristic exceeds the lower bound. The row-epitaxial algorithm achieves an excellent tradeoff between solution quality and runtime (over 13% improvement with a 10× speed-up over the TSP+1-Threading algorithm of Hannenhalli *et al.* (2002) for 500 × 500 arrays). The sliding-window matching has slightly worse solution quality, but has much better scaling runtime than the other methods. Parallelized execution can achieve further speedups and/or permit the use of stronger local improvement operators.

4. IN-PLACE OPTIMIZATION OF PROBE EMBEDDINGS

In our experience, we have found that separate optimization of probe placement and embedding yields better results for AADP than simultaneous optimization. For example, the asynchronous version of the epitaxial algorithm (Kahng *et al.*, 2002) and the asynchronous version of sliding-window matching (Kahng *et al.*, 2003) are both dominated by algorithms that work in two steps:

Step (i). Find a two-dimensional placement based on synchronous embedding for the probes (using, e.g., the row-epitaxial and sliding-window matching algorithms discussed in the previous section, or the TSP+1-Threading of Hannenhalli *et al.* [2002]).
Step (ii). Iteratively optimize probe embeddings, *without changing their location on the array.*

In this section, we consider the second step of the above flow. We begin with algorithms for optimally embedding a single probe with respect to its neighbors. Then, we establish a lower bound on the optimum border cost for in-place probe embedding; this lower bound improves over Theorem 2 by taking into account the constraint on the placement of the probes. Finally, we propose and compare three methods for in-place optimization of probe embeddings, which we call the *Batched Greedy*, the *Chessboard*, and the *Sequential* algorithms.

```
Input: Nucleotide deposition sequence $S = s_1s_2 \ldots s_K$, $s_i \in \{A,C,G,T\}$; set $X$ of probes already embedded into $S$; and
unembedded probe $p = p_1p_2 \ldots p_k$, $p_i \in \{A,C,G,T\}$
Output: The minimum number of conflicts between an embedding of $p$ and probes in $X$, along with a minimum-conflict
embedding

1. For each $j = 1,\ldots,K$, let $x_j$ be the number of probes in $X$ which have a non-blank letter in $j^{th}$ position.
2. $cost(0,0) = 0$; For $i = 1,\ldots,k$, $cost(i,0) = \infty$
3. For $j = 1,\ldots,K$ do
      $cost(0,j) = cost(0,j-1) + x_j$
      For $i = 1,\ldots,k$ do
          If $p_i = s_j$ then $cost(i,j) = \min\{cost(i,j-1) + x_j,\ cost(i-1,j-1) + |X| - x_j\}$
          Else $cost(i,j) = cost(i,j-1) + x_j$
4. Return $cost(k,K)$ and the corresponding embedding of $s$
```

FIG. 6. The Single Probe Alignment Algorithm.

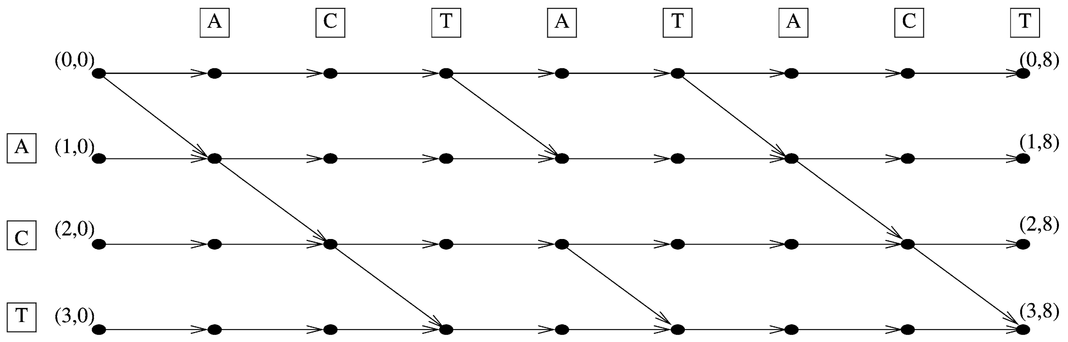

FIG. 7. Directed acyclic graph G_1 representing possible embeddings of probe $p = ACT$ into the nucleotide deposition sequence $S = ACTATACT$.

4.1. Optimum embedding of a single probe

The basic operation used by in-place embedding optimization algorithms (Section 4.3) is to find the optimum embedding of a probe when the adjacent sites contain already-embedded probes. In other words, our goal is to simultaneously align the given probe s to its embedded neighboring probes, while making sure this alignment gives a feasible embedding of s in the nucleotide deposition sequence S. In this section, we give an efficient dynamic programming algorithm for computing this optimum alignment.

The Single Probe Alignment algorithm (see Fig. 6) essentially computes a shortest path in a specific directed acyclic graph $G = (V, E)$. Let p be the probe to be aligned, and let X be the set of already embedded probes adjacent to p. Each embedded probe $q \in X$ is a sequence of length $K = |S|$ over the alphabet $\{A, C, G, T, b\}$, with the j^{th} letter of q being either a blank or s_j, the j^{th} letter of the nucleotide deposition sequence S. The graph G (see Fig. 7) has vertex set $V = \{0, \ldots, k\} \times \{0, \ldots, K\}$ (where k is the length of the probe p and K is the length of the deposition sequence S), and edge set $E = E_{horiz} \cup E_{diag}$ where

$$E_{horiz} = \{(i, j-1) \rightarrow (i, j) \mid 0 \leq i \leq k, 0 < j \leq K\}$$

and

$$E_{diag} = \{(i-1, j-1) \rightarrow (i, j) \mid 0 < i \leq k, 0 < j \leq K, p_i = s_j\}.$$

The cost of a horizontal edge $(i, j-1) \rightarrow (i, j)$ is defined as the number of embedded probes in X which have a nonblank letter on j^{th} position, while the cost of a diagonal edge $(i-1, j-1) \rightarrow (i, j)$ is equal to the number of embedded probes of X with a blank on the j^{th} position. The Single Probe Alignment algorithm computes the shortest path from the source node $(0, 0)$ to the sink node (k, K) using a topological traversal of G (the graph G is not explicitly constructed).

Theorem 3. *The algorithm in Fig. 6 returns, in $O(kK)$ time, the minimum number of conflicts between an embedding of s and the adjacent embedded probes X (along with a minimum-conflict embedding of s).*

Proof. Each directed path from $(0, 0)$ to (k, K) in G consists of K edges, k of which must be diagonal. Each such path P corresponds to an embedding of p into S as follows. If the j^{th} arc of P is horizontal, the embedding has a blank in j^{th} position. Otherwise, the j^{th} arc must be of the form $(i-1, j-1) \rightarrow (i, j)$ for some $1 \leq i \leq k$, and the embedding of p corresponding to P has $p_i = s_j$ in the j^{th} position. It is easy to verify that the edge costs defined above ensure that the total cost of P gives the number of conflicts between the embedding of p corresponding to P and the set X of embedded neighbors. ■

Remark. The above dynamic programming algorithm can be easily extended to find the optimal simultaneous embedding of $n > 1$ probes. The corresponding directed acyclic graph G consist of $k^n K$ nodes $(i_1, \ldots, i_n, j)$, where $0 \leq i_l \leq k$, $1 \leq j \leq K$. All arcs into $(i_1, \ldots, i_n, j)$ come from nodes $(i'_1, \ldots, i'_n, j-1)$, where $i'_l \in \{i_l, i_l - 1\}$. Therefore, the indegree of each node is at most 2^n. The weight of each edge is defined as above such that each finite weight path defines embeddings for all probes and the weight equals the number of conflicts. Finally, computing the shortest path between $(0, \ldots, 0)$ and $(k, \ldots, k, K)$ can be done in $O(2^n k^n K)$ time.

4.2. Lower bounds for in-place embedding optimization

To get an estimate on the how much improvement is possible, let $LG2$ be a grid graph $G2$ with weights on edges equal to the LCS distance between endpoint probes. The following lower bound is obvious.

Theorem 4. *The total edge weight of the graph $LG2$ is a lower bound on the optimum AADP solution cost with a given placement.*

Note. A more accurate lower bound can be obtained by replacing LCS distance with *embedded LCS* distance, $elcsd(p, p')$, which is the minimum number of conflicts over all possible pairs of embeddings of the probes p and p'. The embedded LCS distance can be computed using an $O(|p| \cdot |p'| \cdot |S|)$ dynamic programming algorithm. Unfortunately, neither of these lower bounds is tight, as can be seen from the example in Fig. 3.

4.3. Algorithms for iterative in-place embedding optimization

Batched greedy optimization. We have implemented a natural greedy algorithm (GA) for optimizing probe embeddings. The GA finds a probe that offers largest cost gain from optimum reembedding with respect to the (fixed) embeddings of its neighbors; the algorithm then implements this reembedding, updates gains, and repeats. A faster *batched* version of GA (see Fig. 8) partially sacrifices its greedy nature in favor of runtime, via the mechanism of less-frequent gain updates. In other words, during a single *batched* phase we reembed probes in greedy order according to the cost gains from reembedding, but we do not update any gain while there are still independent probes with positive gain.

Chessboard optimization. The main idea behind our so-called "Chessboard" Algorithm is to maximize the number of *independent* reembeddings, where two probes are independent if changing the embedding of one does not affect the optimum embedding of the other. It is easy to see that if we bicolor our grid as we would a chessboard, then all white (respectively, black) sites will be independent and can therefore be simultaneously, and optimally, reembedded. The Chessboard Algorithm (see Fig. 9) alternates reembeddings of black and white sites until no improvement is obtained.

A 2×1 version of the Chessboard Algorithm partitions the array into iso-oriented 2×1 tiles and bicolors them. Then, using the multi-probe alignment algorithm (see the remark in Section 4.1) with $n = 2$, it alternatively optimizes the black and white 2×1 tiles.

```
Input: Feasible AADP solution, i.e., placement in $G2$ of probes embedded in $S$
Output: A heuristic low-cost feasible AADP solution
------------------------------------------------------------
While there exist probes which can be re-embedded with gain in cost do
    Compute gain of the optimum re-embedding of each probe.
    Unmark all probes
    For each unmarked probe $p$, in descending order of gain, do
        Re-embed $p$ optimally with respect to its four neighbors
        Mark $p$ and all probes in adjacent sites
```

FIG. 8. The Batched Greedy Algorithm.

```
Input: Feasible AADP solution, i.e., placement in $G2$ of probes embedded in $S$
Output: A heuristic low-cost feasible AADP solution
------------------------------------------------------------
Repeat until there is no gain in cost
    For each site $(i,j)$, $1 \leq i,j \leq N$ with $i+j$ even, re-embed probe optimally with respect to its four neighbors
    For each site $(i,j)$, $1 \leq i,j \leq N$ with $i+j$ odd, re-embed probe optimally with respect to its four neighbors
```

FIG. 9. The Chessboard Algorithm.

Sequential optimization. In this method, we perform optimal reembedding of probes in a sequential row-by-row fashion. A shortcoming of the Batched Greedy and Chessboard Algorithms is that, by always reembedding *independent* sets of probes, it takes longer to propagate the effects of a new embedding. Performing the reembedding sequentially permits faster propagation and convergence to a better local optimum.

4.4. *Empirical evaluation of in-place embedding optimization algorithms*

We have implemented the four in-place embedding optimization algorithms discussed in the previous section, namely, the Batched Greedy, Chessboard, 2×1 Chessboard, and Sequential algorithms. To improve the runtime, we stop all algorithms as soon as the improvement for an iteration drops below 0.1% of the total number of conflicts.[5] Table 2 gives the results obtained by the four algorithms when applied to the two-dimensional placement constructed by the TSP+1-Threading algorithm (Hannenhalli *et al.*, 2002).[6] For comparison, we include the lower-bound given by Theorem 4; the columns labeled "%Gap" show by how much the respective heuristic exceeds the lower bound.

The results show that the Chessboard Algorithm is better than Batched Greedy Algorithm with comparable running time. The 2×1 Chessboard improves solution quality by a further 0.8–1.2%, coming within 19–20% of the lower bound for arrays of size between 100×100 and 500×500. We found the Sequential Algorithm to give the best tradeoff between solution quality and run time: its run time is comparable to that of Batched Greedy and Chessboard, yet its solution quality comes close and sometimes exceeds that of 2×1 Chessboard.

[5]In our experiments, imposing the threshold of 0.1% leads to a total loss in solution quality of at most 1%. On the other hand, the number of iterations and hence the runtime decreases by more than one order of magnitude.

[6]Experiments were performed on arrays of 25-mer probes chosen uniformly at random. Run times are for a dual-processor 1.4 GHz Intel Xeon server with 512 MB RAM.

TABLE 2. GAP FROM THE LOWER BOUND GIVEN BY THEOREM 4 AND CPU SECONDS (AVERAGES OVER 10 RANDOM INSTANCES) FOR THE FOUR IN-PLACE EMBEDDING OPTIMIZATION ALGORITHMS

Chip	*Lower bound*	*Batched greedy*		*Chessboard*		*2x1 Chessboard*		*Sequential*	
size	*Cost*	*Gap (%)*	*CPU*	*Gap (%)*	*CPU*	*Gap (%)*	*CPU*	*Gap (%)*	*CPU*
100	364953	25.7	40	20.5	54	19.4	480	19.9	64
200	1425784	26.3	154	20.9	221	19.7	1915	20.3	266
300	3130158	26.7	357	21.5	522	21.6	4349	20.6	577
500	8590793	27.1	943	21.4	1423	20.2	15990	20.9	1535

5. PRACTICAL EXTENSIONS

The following extensions to the border length minimization problem are important in practice, but have not been addressed by previous works on the problem (Hannenhalli *et al.*, 2002).

1. **Distance-dependent border conflict weights.** Back-reflection of light does not affect only adjacent array cells, i.e., cells sharing an edge. To a lesser degree, it also affects cells that share a corner, and even cells that are as far as three cells apart (Hubbell and Mittmann, 2002). This implies that an accurate formulation of the problem should weight conflicts according to the distance between cells.
2. **Position-dependent border conflict weights.** The weight of border conflicts depends on the position in the probe since contamination errors are more harmful in the middle of the probe (Hubbell and Mittmann, 2002). Suggested weights are given by the square root of the distance to the closer endpoint (so conflict weight varies from 1 to $\sqrt{12}$ in a 25-mer).
3. **Polymorphic probes.** Some of the synthesized DNA probes occur both unmodified and mutated in the middle position (e.g., for detection of single-nucleotide polymorphisms—SNPs—in the target DNA or for reliability of the hybridization test). To minimize border length, the SNPs are placed together, so the general BMP requires placing and aligning a mixture of single probes, 2- and 4-ominoes.

Extending most of our methods to handle these extensions is straightforward; here we focus here on the extension of the optimal alignment algorithm given in Section 4.1. Define a *probe* to be a set of 1, 2, or 4 k-mers (SNPs) which differ only in the middle position. We will assume that the SNPs in a probe are always placed in adjacent cells forming 1×1, 2×1, or 2×2 rectangles, respectively. Furthermore, we assume that the SNPs in a probe are always aligned to each other except for the single position where the mutation occurs. Although the optimum solution may not always satisfy these constraints, imposing them should lead only to a very small loss in solution quality.

We will first describe the algorithm for embedding a probe consisting of a single k-mer and then generalize it to the case of two or four SNPs per probe. We use the following notations:

- $S = s_1 \dots s_K$= nucleotide deposition sequence
- k= length of probes (typically $k = 25$)
- $||q||_j$= number of nonblank letters among the first j positions of embedded probe q
- $h(c, c')$= horizontal distance between cells c and c'
- $v(c, c')$= vertical distance between cells c and c'
- $w : N_+ \times N_+ \to [0, 1]$, finite support function[7] giving the conflict weight between a masked cell and an unmasked cell as a function of the horizontal and vertical distances between them
- ω_i, $i = 1, \dots, k$, nonnegative conflict weight multipliers depending on the position of the erroneously inserted nucleotide, e.g., $\omega(i) = \sqrt{\min\{i, k - i\}}$

The embedding algorithm for a probe $p = \{p_1 \dots p_k\}$ with no mutations (Fig. 10) is essentially identical to the the algorithm in Fig. 6 except for the different costs assigned to the edges of the underlying directed acyclic graph G. Let c_p be the array cell assigned to p, and, for every array cell $c \neq c_p$, let q_c be the

[7]The support of a function f is the subset of its domain where f has nonzero values.

```
Input: Nucleotide deposition sequence $S = s_1 s_2 \ldots s_K$, $s_i \in \{A,C,G,T\}$; probe $p = p_1 p_2 \ldots p_k$, $p_i \in \{A,C,G,T\}$, probe
location $c_p$, and probe embeddings $q_c$, $c \neq c_p$
Output: Minimum conflict weight along with a minimum conflict embedding of $p$

1. Compute $x_{ij}$ and $y_{ij}$ for each $i = 1,\ldots,k$ and $j = 1,\ldots,K$ using (1) and (2)
2. $cost(0,0) = 0$; For $i = 1,\ldots,k$, $cost(i,0) = \infty$
3. For $j = 1,\ldots,K$ do
       $cost(0,j) = cost(0,j-1) + x_{ij}$
       For $i = 1,\ldots,k$ do
           If $p_i = s_j$ then $cost(i,j) = \min\{cost(i,j-1) + x_{ij},\ cost(i-1,j-1) + y_{ij}\}$
           Else $cost(i,j) = cost(i,j-1) + x_{ij}$
4. Return $cost(k,K)$ and the corresponding embedding of $p$
```

FIG. 10. The embedding algorithm for a probe with no mutations.

embedded probe placed in c. In other words, every q_c is a sequence of length $K = |S|$ over the alphabet $\{A, C, G, T, b\}$, with the j^{th} letter of q_c being either b (blank) or s_j. We define the cost of a horizontal edge $(i, j-1) \to (i, j)$ to be

$$x_{ij} = \omega(i) \sum_{c \neq c_p,\ (q_c)_j \neq b} w(h(c_p, c), v(c_p, c)) \tag{1}$$

and the cost of a diagonal edge $(i-1, j-1) \to (i, j)$ to be

$$y_{ij} = \sum_{c \neq c_p,\ (q_c)_j = b} \omega(||q_c||_j) w(h(c, c_p), v(c, c_p)). \tag{2}$$

It is easy to verify that the total cost of a path P from $(0, 0)$ to (k, K) equals the weighted number of conflicts between the corresponding embedding of p and the surrounding embedded probes.

Theorem 5. *The algorithm in Fig. 10 returns the minimum conflict weight along with a minimum conflict embedding of p in $O(kK + KW)$ time, where W is the size of the finite support of w.*

Proof. The correctness follows by observing that the algorithm implicitly computes a shortest path from the source node $(0, 0)$ to the sink node (k, K) using a topological traversal of G_1. Since steps 2–4 take $O(kK)$ time, the run time is dominated by computing the $O(kK)$ edge costs in Step 1. Since each x_{ij} is the product of two values, one depending only on i and the other only on j, calculating all values x_{ij} can be done in $O(kK + KW)$ time. Similarly, y_{ij}'s are independent of i and can be computed in $O(kK + KW)$ time overall. ∎

We now consider a probe p consisting of two SNPs, namely, $p_1 \ldots p_{m-1} p_m p_{m+1} \ldots p_k$ and $p_1 \ldots p_{m-1} p'_m p_{m+1} \ldots p_k$. Let c_p and c'_p be two adjacent array cells in which the two SNPs must be placed. Besides embedding the two SNPs into the nucleotide deposition sequence, embedding p also requires deciding which SNP goes into c_p and which one goes into c'_p. Finding the optimum embedding of p can be cast as a shortest path problem in a new directed acyclic graph G_2 (see Fig. 11) obtained from G_1 by the following:

1. Deleting vertices (m, j), $j = 0, \ldots, K$ and the edges incident to them.
2. Changing the cost of each remaining horizontal edge $(i, j-1) \to (i, j)$ to

$$\omega(i) \sum \Big(w(h(c_p, c), v(c_p, c)) + w(h(c'_p, c), v(c'_p, c)) \Big) \tag{3}$$

where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j \neq b$.

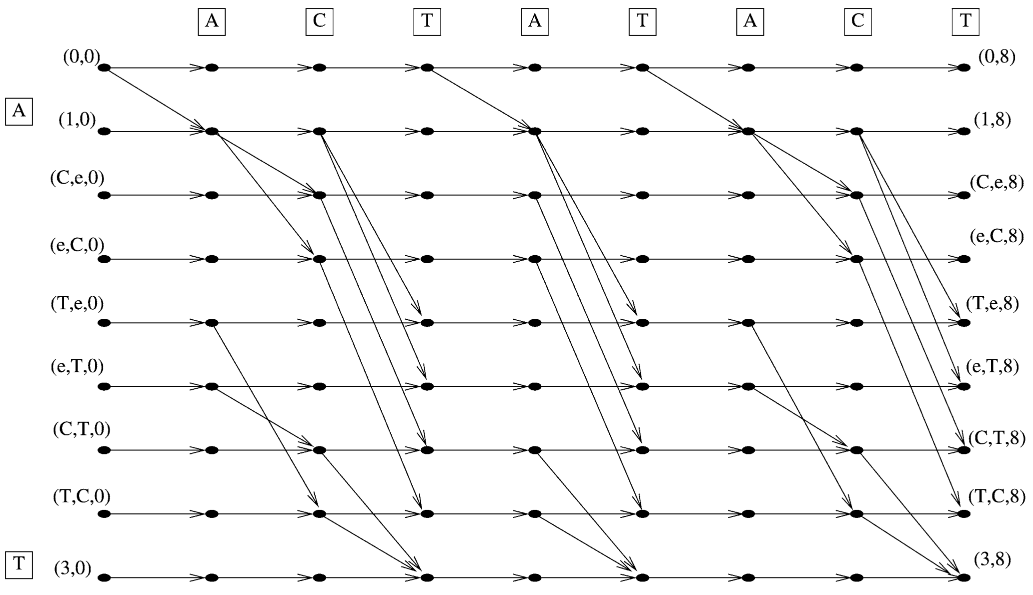

FIG. 11. Directed acyclic graph G_2 representing possible embeddings of probe $p = A\{C|T\}T$ into the nucleotide deposition sequence $S = ACTATACT$.

3. Changing the cost of each remaining diagonal edge $(i-1, j-1) \to (i, j)$ to

$$\omega(||q_c||_j) \sum \Big(w(h(c, c_p), v(c, c_p)) + w(h(c, c'_p), v(c, c'_p)) \Big) \tag{4}$$

   where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j = b$.
4. Inserting $6(K+1)$ new vertices (α, β, j), where $\alpha, \beta \in \{\varepsilon, p_m, p'_m\}$, $\alpha \neq \beta$, and $j = 0, \ldots, K$.
5. Inserting, for every $j = 1, \ldots, K$, horizontal edges $(\alpha, \beta, j-1) \to (\alpha, \beta, j)$ with cost

$$\omega(m - 1 + |\alpha|) \sum w(h(c_p, c), v(c_p, c)) \\ + \omega(m - 1 + |\beta|) \sum w(h(c'_p, c), v(c'_p, c)) \tag{5}$$

   where the sums are taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j \neq b$.
6. Inserting diagonal edges
   - $(m-1, j-1) \to (p_m, \varepsilon, j)$, respectively $(m-1, j-1) \to (p'_m, \varepsilon, j)$, for every j such that $p_m = s_j$, respectively $p'_m = s_j$, each with cost

$$\omega(m-1) w(h(c'_p, c_p), v(c'_p, c_p)) \\ + \omega(||q_c||_j) \sum w(h(c, c_p), v(c, c_p)) \tag{6}$$

     where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j = b$;
   - $(m-1, j-1) \to (\varepsilon, p_m, j)$, respectively $(m-1, j-1) \to (\varepsilon, p'_m, j)$, for every j such that $p_m = s_j$, respectively $p'_m = s_j$, each with cost

$$\omega(m-1) w(h(c_p, c'_p), v(c_p, c'_p)) \\ + \omega(||q_c||_j) \sum w(h(c, c'_p), v(c, c'_p)) \tag{7}$$

     where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j = b$;

- $(\varepsilon, p'_m, j-1) \rightarrow (p_m, p'_m, j)$, respectively $(\varepsilon, p_m, j-1) \rightarrow (p'_m, p_m, j)$,for every j such that $p_m = s_j$, respectively $p'_m = s_j$, each with cost

$$\omega(m)w(h(c'_p, c_p), v(c'_p, c_p)) \\ + \omega(||q_c||_j) \sum w(h(c, c_p), v(c, c_p)) \quad (8)$$

  where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j = b$;
- $(p'_m, \varepsilon, j-1) \rightarrow (p'_m, p_m)$, respectively $(p_m, \varepsilon, j-1) \rightarrow (p_m, p'_m, j)$, for every j such that $p_m = s_j$, respectively $p'_m = s_j$, each with cost

$$\omega(m)w(h(c_p, c'_p), v(c_p, c'_p)) \\ + \omega(||q_c||_j) \sum w(h(c, c'_p), v(c, c'_p)) \quad (9)$$

  where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j = b$;
- $(p_m, p'_m, j-1) \rightarrow (m+1, j)$ and $(p'_m, p_m, j-1) \rightarrow (m+1, j)$, both with cost given by (4), for every j such that $p_{m+1} = s_j$.

The definition of G_2 ensures that each directed path from (0, 0) to (k, K) corresponds to an embedding of the two-SNPs of probe p. Since the costs of the $O(kK)$ edges of G_2 can still be computed in $O(kK+KW)$ time, it follows that the minimum conflict embedding of a two-SNP probe can be computed in $O(kK+KW)$ by an algorithm similar to the one in Fig. 6.

The optimal embedding of a probe with four SNPs can be found by a shortest path computation in a graph that similarly represents all possible assignments of the four SNPs to the four array cells, as well as the possible embeddings of the SNPs into the nucleotide deposition sequence. The graph still contains $O(kK)$ edges, and edge costs can still be computed in $O(kK+KW)$ time. Therefore, we get the following:

Theorem 6. *The minimum conflict embedding of a two- or four-SNP probe can be computed in* $O(kK+KW)$ *time.*

6. EMPIRICAL EVALUATION OF OVERALL AADP FLOWS

We conducted experiments on both random and industry datasets to compare our AADP flows. Table 3 gives results for our flows performing optimization of probe placement followed by in-place optimization of probe embeddings. As in previous sections, these experiments were run on arrays of 25-mer probes chosen uniformly at random. In these experiments, array size was varied between 100×100 and 1,000×1,000. The results indicate that our methods are highly scalable and yield high-quality solutions for AADP. The best solution quality is achieved by Row-Epitaxial followed by Sequential alignment: this flow improves by up to 35% over the synchronous placements computed using the TSP+1Threading method of Hannenhalli *et al.* (2002) and up to 10% over the method of Hannenhalli *et al.* (2002) followed by Sequential alignment. Sliding-Window Matching has the best scaling run time, completing a million-probe placement in tens of minutes of CPU time on a 1.4 GHz Intel Xeon server with 512 MB RAM, while still beating in solution quality the TSP+1Threading algorithm followed by Sequential alignment.

For comparison, we included in Table 3 the lower-bound given by Theorem 4. While the gap to the lower bound tends to increase with chip size for all compared algorithms, we do notice some very positive trends for the border cost normalized by the number of pairs of adjacent array sites, i.e., the *average* number of conflicts per pair of adjacent sites. For probes of length 25, as were those used in our experiments, the maximum normalized cost is 50 (when there is a complete mismatch between adjacent probes). For any array size, random placement of synchronously embedded random probes leads to an expected normalized cost of 37.5, since one expects matches between two adjacent probes for one quarter of the nucleotides. In contrast, for all the considered AADP algorithms, the normalized cost decreases with increasing chip size, which can be attributed to greater freedom of choice that the algorithms can exploit for the larger chip

TABLE 3. TOTAL BORDER COST, BORDER COST NORMALIZED BY THE NUMBER OF PAIRS OF ADJACENT ARRAY CELLS, AND CPU TIME OF THE COMPARED AADP HEURISTICS (AVERAGES OVER 10 RANDOM INSTANCES)[a]

Chip	*Lower bound*		*TSP+1Threading*[b]		*TSP+1Threading+Seq.*			*Row-Epit.+Seq.*			*SWM+Seq.*		
size	*Cost*	*Norm.*	*Cost*	*Norm.*	*Cost*	*Norm.*	*CPU*	*Cost*	*Norm.*	*CPU*	*Cost*	*Norm.*	*CPU*
100	220497	11.1	554849	28.0	439829	22.2	113	413158	20.9	118	433274	21.9	1
200	798708	10.0	2140903	26.9	1723352	21.6	1901	1593146	20.0	493	1693658	21.2	46
300	—	—	4667882	26.0	3801765	21.2	12028	3503526	19.5	1562	3746722	20.9	112
500	—	—	12702474	25.5	10426237	20.9	109648	9418042	18.9	8400	10049442	20.1	302
1000	—	—	—	—	—	—	—	35918568	17.9	41740	38898792	19.5	1307

[a]We used 6×6 windows with overlap 3 for SWM, and $k_0 = 20$ look-ahead rows for row-epitaxial.
[b]Hannenhalli *et al.*, 2002.

sizes. The row-epitaxial-based flow results in a normalized cost of 17.9 for arrays of size 1,000$\times$1,000, which represents an improvement of 52% over the expected normalized cost of a random placement.

We have also validated our methods on the set of probes for an Affymetrix Humane Genome chip. This 712×712 DNA chip is filled by pairs of SNP's (each consisting of the original probe and a copy with the middle nucleotide changed) and small amount of control probes ($< 1\%$) each having a predetermined placement. The (truncated) periodic nucleotide deposition sequence used by Affymetrix (and in our experiments) has length 74: this sequence is sufficiently long to accommodate all probes and is cheaper than the universal 100-long nucleotide sequence by 26%. The flow that gave the best results consists of the following steps: (1) probe embedding using a SNP-aware version of "earliest possible" embedding; (2) lexicographical sorting of the embedded probes followed by 1-threading; (3) synchronous sliding window matching—we used 48$\times$48 windows with overlap 24—where synchronous here refers to computing Hamming distances between embedded probes rather than unembedded probes; (4) row-epitaxial placement with $k_0 = 80$ rows of look-ahead; and (5) a SNP-aware version of the Sequential alignment algorithm. The final number of conflicts was reduced by 4.2% with respect to the Affymetrix optimized placement.[8]

7. CONCLUSIONS

In this paper, we have studied DNA array design problems which seek to minimize the unintended illumination during manufacturing. We have suggested highly scalable algorithms for synchronous probe placement and for in-place probe embedding optimization. Combining these algorithms yields better and faster solutions to the DNA array design problem compared to previous methods; our approach also compares favorably to industry placements.

We are currently optimizing our implementation for multiprocessing and more practical cost criteria. Here, other metaheuristic frameworks (e.g., large-step Markov chains) are of interest, as are potential improvements that our discussion has already noted. We also seek to integrate probe selection and array reliability aspects into our placement and embedding problem formulation. Developing tighter lower bounds and other means of assessing sub-optimality of array design algorithms is another important direction for further work.

ACKNOWLEDGMENTS

The authors wish to thank Earl Hubbell and Michael Mittman for providing many relevant details of the problem and the Human Genome chip data used in the experimental study. The work of A.B.K., I.I.M., and S.R. was partially supported by Cadence Design Systems, Inc. and the MARCO Gigascale Silicon Research Center. The work of P.A.P. was partially supported by the Ronald R. Taylor chair fund. The work of A.Z.Z. was partially supported by NIH Award # 1 P20 GM065762-01A1, NSF Grant CCR-9988331, Award No. MM2-3018 of MRDA and CRDF, and by the State of Georgia's Yamacraw Initiative.

REFERENCES

Abdueva, D., and Skvortsov, D. *Personal communication* (University of Southern California, Los Angeles, CA).

Akers, S. 1981. On the use of the linear assignment algorithm in module placement. *Proc. 1981 ACM/IEEE Design Automation Conference (DAC'81)*, 137–144.

Alon, N., Colbourn, C.J., Lingi, A.C.H., and Tompa, M. 2001. Equireplicate balanced binary codes for oligo arrays. *SIAM J. Dis. Math.* 14, 481–497.

[8]We understand (Hubbell and Mittmann, 2002) that Affymetrix uses placement techniques that are similar to row-epitaxial placement combined with earliest possible alignment. This probably explains why our improvement is relatively small. Furthermore, we note that the reduction in number of masks from 100 to 74 also reduces somehow the freedom that can be exploited by our dynamic programming alignment algorithms.

Alpert, C.J., and Kahng, A.B. 1994. Multi-way partitioning via spacefilling curves and dynamic programming. *Proc. 1994 ACM/IEEE Design Automation Conference (DAC'94)*, 652–657.

Bartholdi, J.J., and Platzman, L.K. 1982. An O(N log N) planar travelling salesman heuristic based on spacefilling curves. *Operations Res. Lett.* 1, 121–125.

Caldwell, A.E., Kahng, A.B., and Markov, I.L. 1999. Optimal partitioners and end-case placers for standard-cell layout. *Proc. ACM 1999 Int. Symposium on Physical Design (ISPD'99)*, 90–96.

Doll, K., Johannes, F.M., and Antreich, K.J. 1994. Iterative placement improvement by network flow methods. *IEEE Trans. Computer-Aided Design* 13, 1189–1200.

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

Hannenhalli, S., Hubbell, E., Lipshutz, R., and Pevzner, P.A. 2002. Combinatorial algorithms for design of DNA arrays, *in Chip Technology*, Springer-Verlag, NY.

Huang, D.J., and Kahng, A.B. 1997. Partitioning-based standard cell global placement with an exact objective. *Proc. 1997 ACM Int. Symp. Physical Design (ISPD'97)*, 18–25.

Hubbell, E., and Mittmann, M. 2002. *Personal communication* (Affymetrix, Santa Clara, CA), July 2002.

Hubbell, E., and Pevzner, P.A. 1999. Fidelity probes for DNA arrays. *Proc. 1999 Int. Conf. Intelligent Systems for Molecular Biology (ISMB'99)*, 113–117.

Kahng, A.B., Măndoiu, I.I., Pevzner, P.A., Reda, S., and Zelikovsky, A. 2002. Border length minimization in DNA array design. *Proc. 2002 Int. Workshop on Algorithms in Bioinformatics (WABI'02)*, 435–448.

Kahng, A.B., Măndoiu, I.I., Pevzner, P.A., Reda, S., and Zelikovsky, A. 2003. Engineering a scalable placement heuristic for DNA probe arrays. *Proc. 2003 Int. Conf. Research in Computational Molecular Biology (RECOMB'03)*, 148–156.

Kasif, S., Weng, Z., Derti, A., Beigel, R., and DeLisi, C. 2002. A computational framework for optimal masking in the synthesis of oligonucleotide microarrays. *Nucl. Acids Res.* 30, e106.

Li, F., and Stormo, G.D. 2000. Selecting optimum DNA oligos for microarrays. *Proc. 2000 IEEE Int. Symp. Bio-Informatics and Biomedical Engineering (BIBE'00)*, 200–207.

Lipshutz, R.J., Fodor, S.P., Gingeras, T.R., and Lockhart, D.J. 1999. High density synthetic oligonucleotide arrays. *Nature Genet.* 21, 20–24.

Preas, B.T., and Lorenzetti, M.J. eds. 1988. *Physical Design Automation of VLSI Systems*, Benjamin-Cummings, Menlo Park, CA.

Rahmann, S. 2002. Rapid large-scale oligonucleotide selection for microarrays. *Proc. 2002 IEEE Computer Society Bioinformatics Conference (CSB'02)*, 54–63.

Rahmann, S. 2003. The shortest common supersequence problem in a microarray production setting. *Bioinformatics* 19(suppl. 2), 156–161.

Sengupta, R., and Tompa, M. 2002. Quality control in manufacturing oligo arrays: A combinatorial design approach. *J. Comp. Biol.* 9, 1–22.

Shahookar, K., and Mazumder, P. 1991. VLSI cell placement techniques. *Computing Surveys* 23, 143–220.

Steinberg, L. 1961. The backboard wiring problem: A placement algorithm. *SIAM Review* 3, 37–50.

Tolonen, A.C., Albeanu, D.F., Corbett, J.F., Handley, H., Henson, C., and Malik, P. 2002. Optimized in situ construction of oligomers on an array surface. *Nucl. Acids Res.* 30, e107.

Address correspondence to:
Ion I. Măndoiu
Computer Science and Engineering Department
University of Connecticut
371 Fairfield Road, Unit 1155
Storrs, CT 06269-3155

E-mail: ion@engr.uconn.edu