Border Length Minimization in DNA Array Design*

A.B. Kahng[1], I.I. Măndoiu[1], P.A. Pevzner[1], S. Reda[1], and A.Z. Zelikovsky[2]

[1] Department of Computer Science & Engineering
University of California at San Diego, La Jolla, CA 92093
{abk,mandoiu,ppevzner,sreda}@cs.ucsd.edu
[2] Department of Computer Science
Georgia State University, Atlanta, GA 30303
alexz@cs.gsu.edu

Abstract. Optimal design of DNA arrays for very large-scale immobilized polymer synthesis (VLSIPS) [3] seeks to minimize effects of unintended illumination during mask exposure steps. Hannenhalli et al. [6] 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. Their method reduces the number of conflicts, i.e., total mask border length, by 20-30% versus the previous standard method for array design. In this paper, we propose a probe placement algorithm for synchronous array synthesis which reduces the number of conflicts by up to 10% versus the method of Hannenhalli et al [6]. We also consider the case of *asynchronous* array synthesis, and present new heuristics that reduce the number of conflicts by up to a further 15.5-21.8%. Finally, we give lower bounds that offer insights into the amount of available further improvements. The paper concludes with several directions for future research.

1 Introduction

DNA *probe arrays* are used in a wide range of genomic analyses. As described in [3], during very large-scale immobilized polymer synthesis (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 consisting of nontransparent and

* The work of ABK, IIM, and SR was partially supported by Cadence Design Systems, Inc. and the MARCO Gigascale Silicon Research Center, the work of PAP was supported by Ronald R. Taylor chair fund and the work of AZZ was partially supported by NSF Grant CCR-9988331, Award No. MM2-3018 of the Moldovan Research and Development Association (MRDA) and the U.S. Civilian Research & Development Foundation for the Independent States of the Former Soviet Union (CRDF), and by the State of Georgia's Yamacraw Initiative.

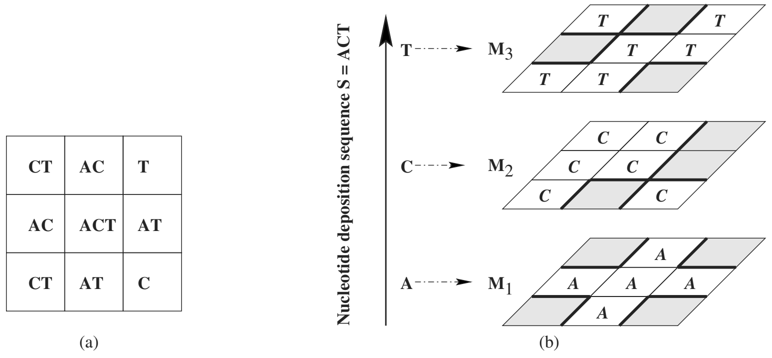

Fig. 1. (a) 2-dimensional probe placement. (b) 3-dimensional probe embedding: the nucleotide deposition sequence $S = (ACT)$ corresponds to the sequence of three masks M_1, M_2 and M_3. In each mask the masked sites are shaded and the borders between exposed and masked sites are thickened.

transparent areas corresponding to the masked and exposed array sites. Each mask induces deposition of a particular nucleotide $s_i \in \{A, C, T, G\})$ at its exposed array sites. The *nucleotide deposition sequence* $S = s_1 s_2 \dots s_K$ corresponding to the sequence of masks is therefore a supersequence of all probe sequences in the array (see Figure 1). 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. The design of DNA arrays raises a number of combinatorial problems that have been discussed by Hubbell and Pevzner [5], Alon et al. [2], Heath and Preparata [4], Li and Stormo [8], Sengupta and Tompa [10], etc. In this paper, we study the *Border Minimization Problem* that was recently introduced by Hannenhalli et al. [6].

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 freedom in how probes are assigned to array sites. The *Border Minimization Problem (BMP)* [6] 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. We may view the array design as a *three-dimensional placement problem* (see Figure 1): two dimensions are represented by the site array, and the third dimension is represented by the sequence S. Each layer in the third dimension corresponds to a mask inducing deposition of a particular nucleotide (A, C, G, or T), and a probe is a "column" within this three-dimensional placement representation. Border length of a given mask is computed as the number of *conflicts*, i.e., pairs of adjacent exposed 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

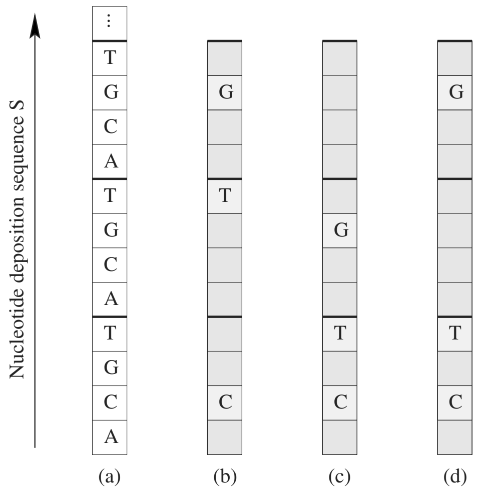

Fig. 2. (a) Periodic nucleotide deposition sequence S. (b) Synchronous embedding of probe CTG into S; the shaded sites denote the masked sites in the corresponding masks. (c-d) Two different asynchronous embeddings of the same probe.

columns. The border length of the embedding 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 (the i^{th}) nucleotide in each probe. This corresponds to a unique (and trivially computed) embedding of each probe p in the sequence S; see Figure 2(b). On the other hand, *asynchronous* array synthesis permits arbitrary embeddings, as shown in Figure 2(c-d).

In the remainder of this paper, we make the following contributions.

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 in [6], 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. [6] enhanced this threading approach to achieve up to 20% border length reduction for large chips. In Section 3, we suggest an *epitaxial* placement heuristic which places a random probe in the center of the array and then continues to insert probes in sites adjacent to already-placed probes, so as to greedily minimize the number of induced conflicts[1]. Our epitax-

[1] Recently a similar heuristic has been explored by Abdueva and Skvortsov [1].

ial placement method improves over the previous TSP-based approach by up to 10%.

Locally Optimal Asynchronous 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 embed a given probe optimally with respect to fixed embeddings of the probe's neighbors. This dynamic programming algorithm is useful whenever we need to improve a given embedding.

Post-placement Improvement of Asynchronous Embeddings. In Section 5 we suggest two ways to improve the embedding of probes that have already been placed (e.g., by threading as in [6], or by epitaxial methods) onto array sites. We describe the *chessboard* and *batched greedy* methods for optimizing probe embeddings *after* assignment of probes to their positions in the array. The *chessboard* method alternates between optimal embedding adjustment of "black" and "white" sites with respect to their neighbors (which always have different color). The *batched greedy* method implements non-conflicting optimal embeddings in a greedy fashion. Our experiments show that the chessboard method (winning slightly over the batched greedy method) decreases by 15.5-21.8% the number of conflicts in the original synchronous placement.

Lower Bounds for the Border Minimization Problem. 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 Subsequence (LCS). These afford an estimate of potential future progress due to improved probe placement algorithms for synchronous embedding. In Section 5 a different LCS-distance based lower bound is applied to the probe embedding, yielding bounds on possible improvement from exploiting this degree of freedom.

2 Array Design Problem Formulations and Lower Bounds

In this section we give a graph-theoretical definition of probe placement and then formulate the synchronous and asynchronous array design problems. For both formulations we give lower bounds on the cost of optimum solutions.

Following the development in [6], 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 $\phi : V_2 \rightarrow V_1$ 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.

In synchronous array design, the Border Minimization Problem (BMP) corresponds to the optimal placement problem for appropriately defined graphs. Let $G2$ be a *2-dimensional grid graph* corresponding to the arrangement of sites in the DNA array. The vertices of $G2$ correspond to $N \times N$ array sites, and edges of $G2$ have weight 1 if endpoints correspond to adjacent sites and 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.

Synchronous Array Design Problem (SADP). Find a minimum-cost placement of the Hamming graph H on the 2-dimensional grid graph $G2$.

Let L be the directed graph over the set of probes obtained by including arcs from each probe to the 4 closest probes with respect to Hamming distance, and then deleting the heaviest $4N$ arcs. Since the graph L has weight no more than the conflict cost of any placement of the set of probes 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, the BMP is more involved. The asynchronous design consists of two steps: (i) the embedding e of each probe p into the nucleotide deposition sequence S, i.e., $e : p \rightarrow S$ such that for each nucleotide $x \in p$, $e(x) = x$ and (ii) the placement of embedded probes into the $N \times N$ array of sites. An embedded probe p may be viewed as a sequence obtained from S by replacing $|S| - |p|$ nucleotides with blanks. Let the *Hamming graph H'* be the graph with vertices corresponding to embedded probes and with edge weights equal to the Hamming distance (we add 1 to the distance between two embedded probes for each nucleotide in one that corresponds to a blank in the other).

Asynchronous Array Design Problem (AADP). Find an embedding of each given probe, and a placement, which minimizes the cost of the placement of the Hamming graph H' on the 2-dimensional grid graph $G2$.

To estimate the cost of the optimum AADP solution, it is necessary to estimate the distance between two probes independent of their embedding into S. Observe that the number of nucleotides (mask steps) common to two embedded probes cannot exceed the length of the *longest common subsequence* (LCS) of these two probes. Define the LCS distance between probes p and p' as $lcsd(p, p') = k - |LCS(p, p')|$, where $k = |p| = |p'|$. Let L' be the directed graph over the set of probes obtained by including arcs from each probe to the 4 closest probes with respect to LCS distance, and then deleting the heaviest $4N$ arcs. Similar to Theorem 1 we get:

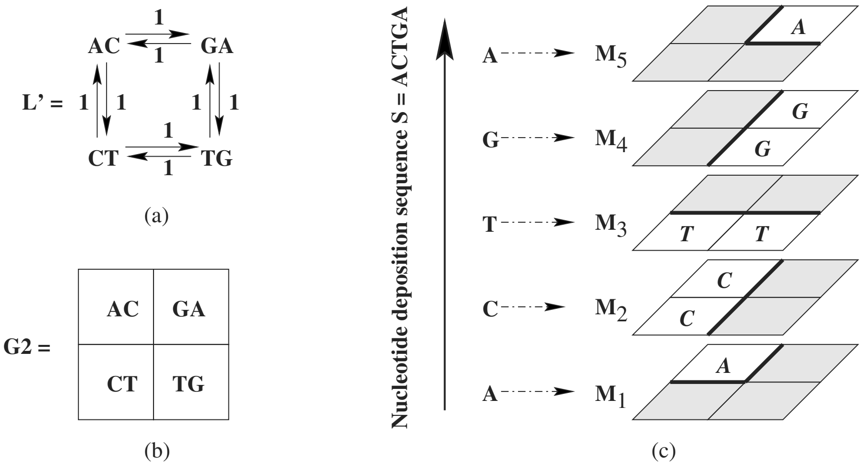

Fig. 3. (a) Lower-bound digraph L' for four probes AC, GA, CT and TG. The lower bound on the number of conflicts is 8 which is the arc weight of L'. (b) Optimum 2-dimensional placement of the probes. (c) Optimum embedding of the probes into the nucleotide deposition supersequence $S = ACTGA$. The optimum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

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

Example. Note that weight of the graph L' may be much smaller than the optimum cost, since in addition to aligning with each other the probes should also be aligned with the nucleotide deposition sequence S. Figure 3 illustrates a case of four 2-nucleotide sequences CT, GC, AG, and TA which should be placed on a 2×2 grid. The lower bound on the number of conflicts is 8 while the optimum number of conflicts is 10.

3 Epitaxial Algorithm for SADP

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 [9,11]. The technique essentially grows a 2-dimensional placement around a single starting "seed".

The intuition behind our use of epitaxial placement is that if more information (i.e., about placed neighboring probes) is available when a site is about to be filled, we can make a better choice of the probe to be placed. For example, if we already know all four neighboring probes for a given site, then our choice of a probe to be placed in this site will be the "best possible", i.e., it will guarantee minimum sum of conflict distances to the four neighbors. The TSP method

```
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.

that has been previously used for DNA array synthesis[2] essentially chooses each probe in the tour based on knowledge about a single neighbor. In contrast, in epitaxial placement an average of 2 neighbors are already known when choosing the probe to be assigned to a site.

The epitaxial algorithm (see Figure 4) places a random probe in 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 the 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 4 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.

4 Optimum Probe Alignment Algorithms

In this section we first give an efficient dynamic programming algorithm for computing the optimal embedding of a single probe whose neighbors are already embedded. We then generalize this algorithm for optimal simultaneous alignment several neighboring probes with respect to already embedded neighbors.

[2] Recall that the method of Hannenhalli et al. [6] first finds a TSP-tour and then threads it into the array. See Section 6 for more details.

Input: Nucleotide deposition sequence $S = s_1 s_2 \ldots s_K$, $s_i \in \{A, C, G, T\}$; set X of probes already embedded into S; and unembedded probe $p = p_1 p_2 \ldots p_k$, $p_i \in \{A, C, G, T\}$
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 blank
      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
       $cost(0,j) = cost(0,j-1) + |X| - x_j$
       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. 5. The Single Probe Alignment Algorithm.

4.1 Optimum Single Probe Alignment Algorithm

The basic operation that we use in the post-placement improvement algorithms (Section 5) finds an optimum embedding of a probe when some (or all) 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 subsection we give an efficient dynamic programming algorithm for computing an optimum alignment.

The Single Probe Alignment algorithm (see Figure 5) 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 Figure 6) is a directed "tilted mesh" with vertex set $V = \{0, \ldots, k\} \times \{0, \ldots, K\}$ (where k is the length of p) and edge set $E = E_{horiz} \cup E_{diag}$ where

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

and

$$E_{diag} = \{(i-1, j-1) \to (i, j) \mid 0 < i \le k, 0 < j \le K\}.$$

The cost of a "horizontal" edge $(i, j-1) \to (i, j)$ is defined as the number of embedded probes in X which have a non-blank letter on j^{th} position, while the cost of a "diagonal" edge $(i-1, j-1) \to (i, j)$ is infinity if the i^{th} letter of p differs from the j^{th} letter of S, and is equal to the number of embedded probes of X with a blank on the j^{th} position otherwise. The Single Probe Alignment algorithm computes the shortest path from the source node $(0, 0)$ to the sink

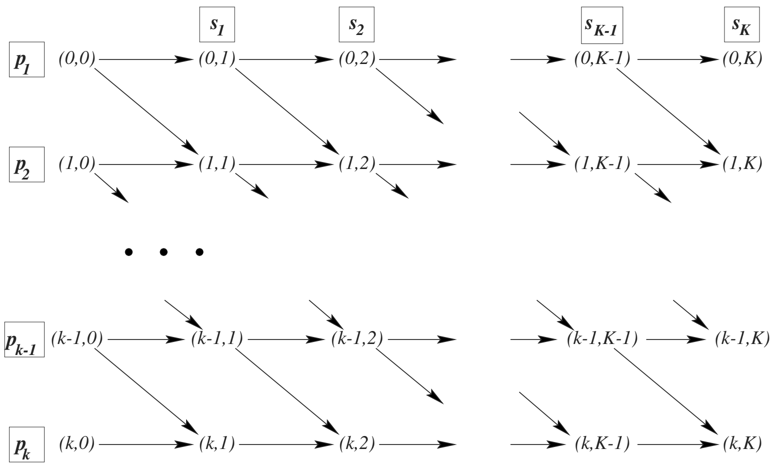

Fig. 6. Directed acyclic graph representing all possible embeddings of probe $p = p_1p_2 \dots p_k$ into mask sequence $S = s_1s_2 \dots s_K$.

node (k, K) using a topological traversal of G (the graph G is not explicitly constructed).

Theorem 3. *The algorithm in Figure 5 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 $(i-1, j-1) \to (i, j)$ for some $1 \le i \le k$, 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.

4.2 Optimum Multiple Probe Alignment Algorithm

The dynamic programming algorithm for the optimal simultaneous embedding of $n > 1$ probes (see [7]) is a straightforward generalization of the algorithm above. The corresponding directed acyclic graph G consist of $k^n K$ nodes $(i_1, \dots, i_n, j)$, where $0 \le i_l \le k$, $1 \le j \le K$. All arcs into $(i_1, \dots, i_n, j)$ come from nodes $(i'_1, \dots, i'_n, j-1)$, 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 n probes and the weight equals the number of conflicts. Finally, computing the shortest path between $(0, \dots, 0)$ and $(k, \dots, k, K)$ can be done in $O(2^n k^n K)$ time.

5 Post-placement Optimization of Probe Embeddings

We now consider an AADP solution flow that is split into two consecutive steps:

Step (i). Find an initial placement and embedding (using e.g., Epitaxial Algorithm (see Figure 4) or TSP+1-Threading), and
Step (ii). Optimize each probe's embedding with respect to its neighbors using the Optimal Alignment Algorithm (see Figure 5).

Theorems 1 and 2 above provide pre-placement lower bounds on the total border (conflict) cost. In this section, we first estimate the limitations of *post-placement* optimization in Step (ii). We then propose two methods of reoptimizing probe embeddings, which we call the *Batched Greedy* and the *Chessboard* algorithms.

5.1 Post-placement Lower Bounds

Once a 2-dimensional placement is fixed, we do not have any freedom left if probes must be embedded synchronously. In the asynchronous array design we can still reembed all probes so as to substantially improve the border length. To get an estimate on the how much improvement is possible, let $LG2$ be a grid graph $G2$ with weights on edges equal to LCS distances 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 (see [7]). Unfortunately, neither of these lower bounds is tight, as can be seen from the example given in Section 2.

5.2 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 Figure 7) 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 probes with positive unchanged gains.

```
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 reembedded with gain in cost do
    Compute gain of the optimum reembedding of each probe.
    Unmark all probes
    For each unmarked probe $p$, in descending order of gain, do
        Reembed $p$ optimally with respect to its four neighbors
        Mark $p$ and all probes in adjacent sites
```

Fig. 7. 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, reembed probe optimally
        with respect to its four neighbors
    For each site $(i, j)$, $1 \leq i, j \leq N$ with $i + j$ odd, reembed probe optimally
        with respect to its four neighbors
```

Fig. 8. The Chessboard Algorithm.

5.3 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 (resp. black) sites will be independent and can therefore be simultaneously, and optimally, reembedded. The Chessboard Algorithm (see Figure 8) 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 multiprobe alignment algorithm (see Section 4.2) with $n = 2$ it alternatively optimizes the black and white 2×1 tiles.

6 Experimental Results

We have implemented two placement algorithms: the multi-fragment Greedy TSP algorithm followed by 1-Threading (see Figure 9) as described in [6] and the Epitaxial algorithm. We have also implemented three post-placement optimization algorithms: Batched Greedy, Chessboard and 2×1 Chessboard. To improve the runtime in our implementations of post-placement algorithms we stop as soon as the iteration improvement drops below 0.1% of the total num-

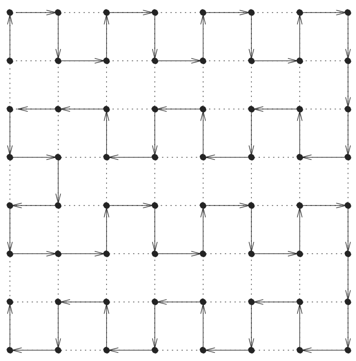

Fig. 9. 1-Threading.

Table 1. Placement heuristics and lower bounds.

Chip	Sync LB	TSP+1-Threading			Epitaxial			Async LB	
Size	Cost	Cost	%Gap	CPU	Cost	%Gap	CPU	Cost	%Gap
20	19242	23401	21.6	0	22059	14.6	0	10236	-46.8
40	72240	92267	27.7	3	85371	18.2	6	38414	-46.8
60	156149	204388	30.9	15	187597	20.1	32	83132	-46.8
80	268525	358945	33.7	46	327924	22.1	104	144442	-46.2
100	410019	554849	35.3	113	505442	23.3	274	220497	-46.2
200	1512014	2140903	41.6	1901	1924085	27.3	4441	798708	-47.2
300	3233861	4667882	44.3	12028	—	—	—	—	—
500	8459958	12702474	50.1	109648	—	—	—	—	—

ber of conflicts[3]. For comparison, we include the synchronous, resp. asynchronous pre-placement lower-bounds given by Theorems 1 and 2, and the post-placement lower-bound given by Theorem 4.

All algorithms were coded in C and compiled using `g++` version 2.95.3 with `-O3` optimization. All reported times are in CPU seconds. Placement algorithms were run on an SGI Origin 2000 with 16 195MHz 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. Post-placement algorithms were run on a dual-processor 1.4GHz Intel Xeon server with 512MB RAM.

All experiments were performed on DNA arrays of size varying between 20×20 to 500×500. All results reported in this section are averages over 10 sets of probes (of length 25, generated uniformly at random).

Table 1 compares the cost (number of conflicts) and runtime of the two placement algorithms. Missing entries were not computed due to prohibitively large running time or memory requirements. For comparison we include the

[3] 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.

Table 2. Optimization of probe embeddings after to the TSP+1-Threading placement.

Chip	LCS LB	Initial	Batched Greedy			Chessboard			2×1 Chessboard		
Size	cost	%Gap	%Gap	#Iter	CPU	%Gap	#Iter	CPU	%Gap	#Iter	CPU
20	14859	57.5	24.3	8.8	2	19.1	13.3	2	18.0	11.1	18
40	59500	55.1	25.0	8.9	7	20.0	12.7	9	18.8	11.8	77
60	133165	53.5	25.3	8.5	15	20.2	13.2	20	19.0	11.9	175
80	234800	52.9	25.7	8.7	27	20.5	13.4	35	19.3	12.0	315
100	364953	52.0	25.7	8.8	40	20.5	13.6	54	19.4	11.7	480
200	1425784	50.2	26.3	8.3	154	20.9	13.9	221	19.7	11.8	1915
300	3130158	49.1	26.7	8.0	357	21.5	13.6	522	21.6	11.0	4349
500	8590793	47.9	27.1	8.0	943	21.4	14.0	1423	20.2	12.0	15990

Table 3. Optimization of probe embeddings after the Epitaxial placement.

Chip	LCS LB	Initial	Batched Greedy			Chessboard			2×1 Chessboard		
Size	cost	%Gap	%Gap	#Iter	CPU	%Gap	#Iter	CPU	%Gap	#Iter	CPU
20	14636	50.7	22.7	8.4	1	17.9	12.3	2	17.0	10.5	17
40	58494	45.9	22.4	8.1	6	17.7	12.9	8	16.8	11.4	74
60	130636	43.6	22.0	8.0	14	17.5	12.3	18	16.6	11.2	164
80	230586	42.2	21.7	8.0	24	17.2	12.6	33	16.3	11.1	289
100	357800	41.3	21.6	8.0	37	17.1	12.1	48	16.3	11.3	461
200	1395969	37.8	20.8	7.1	130	16.4	12.0	190	15.6	10.7	1779

synchronous (resp. asynchronous) placement lower-bounds given by Theorems 1 and 2. The column "%Gap" shows by how much respective heuristic or lower bound exceeds the lower bound for synchronous placement given by Theorem 1. For 200 × 200 DNA arrays, the Epitaxial algorithm improves by more than 10% over the previously best TSP+1-Threading algorithm of [6], coming within 14-27% of the lower-bound given by Theorem 1.

Tables 2–3 give the results of post-placement probe-embedding heuristics when applied to the TSP+1-Threading and Epitaxial placements, respectively. The column "Gap" shows by how much the initial placement and respective heuristics exceed the post-placement LCS lower bound given by Theorem 4. The results show that the Chessboard algorithm is better than Batched Greedy with comparable running time. 2 × 1 Chessboard is giving a further improvement of .8-1.2%, coming within 18-20% and 15-17% of the LCS lower-bound for TSPthreading and Epitaxial placements, respectively. Overall, the best conflict reduction is obtained by applying 2×1 Chessboard optimization to the Epitaxial placement improving [6] by, e.g., 25% for 100x100 chip.

7 Conclusions

For synchronous DNA array design, we have suggested a new epitaxial heuristic which substantially decreases unintended illumination as measured by total border length, during mask exposure steps. We have explored advantages of asynchronous DNA array design, giving several new heuristics to reduce the mask border length. Further directions that we address in ongoing research include:

– developing techniques for simultaneous placement and asynchronous embedding;
– improving solution quality and runtime by incorporating methods such as hierarchical placement (this and several other promising approaches have been previously proposed in the VLSI CAD context); and
– working with industry to assess the quality of our methods on real VLSIPS test cases.

References

1. Abdueva and Skvortsov, *Personal Communication.*
2. N. Alon, C. J. Colbourn, A. C. H. Lingi and M. Tompa, "Equireplicate Balanced Binary Codes for Oligo Arrays", *SIAM Journal on Discrete Mathematics* 14(4) (2001), pp. 481-497.
3. S. Fodor, J. L. Read, M. C. Pirrung, L. Stryer, L. A. Tsai and D. Solas, "Light-Directed, Spatially Addressable Parallel Chemical Synthesis", *Science* 251 (1991), pp. 767-773.
4. S. A. Heath and F. P. Preparata, "Enhanced Sequence Reconstruction With DNA Microarray Application", *Proc. 7th Annual International Conf. on Computing and Combinatoris (COCOON)*, Springer-Verlag, Lecture Notes in Computer Science vol. 2108, August 2001, pp. 64-74.
5. E. Hubbell and P. A. Pevzner, "Fidelity Probes for DNA Arrays", *Proc. Seventh International Conference on Intelligent Systems for Molecular Biology*, 1999, pp. 113-117.
6. S. Hannenhalli, E. Hubbell, R. Lipshutz and P. A. Pevzner, "Combinatorial Algorithms for Design of DNA Arrays", in *Chip Technology* (ed. J. Hoheisel), Springer-Verlag, 2002.
7. A.B. Kahng, I.I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky, "Border Length Minimization in DNA Array Design", Technical Report CS2002-0713, Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA, 2002.
8. F. Li and G.D. Stormo, "Selecting Optimum DNA Oligos for Microarrays", *Proc. IEEE International Symposium on Bio-Informatics and Biomedical Engineering*, 2000, pp. 200-207.
9. B. T. Preas and M. J. Lorenzetti, eds., *Physical Design Automation of VLSI Systems*, Benjamin-Cummings, 1988.
10. R. Sengupta and M. Tompa, "Quality Control in Manufacturing Oligo Arrays: a Combinatorial Design Approach", *Journal of Computational Biology* 9 (2002), pp. 1–22.
11. K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques", *Computing Surveys* 23(2) (1991), pp. 143-220.