Engineering a Scalable Placement Heuristic for DNA Probe Arrays*

A.B. Kahng, I. Măndoiu, P. Pevzner, S. Reda
Department of Computer Science & Engineering
University of California at San Diego
La Jolla, CA 92093
{abk,mandoiu,ppevzner,sreda}@cs.ucsd.edu

A. Zelikovsky
Department of Computer Science
Georgia State University
Atlanta, GA 30303
alexz@cs.gsu.edu

ABSTRACT

Design of DNA arrays for very large-scale immobilized polymer synthesis (VLSIPS) [8] seeks to minimize effects of unintended illumination during mask exposure steps. [9, 14] formulate this requirement as the Border Minimization Problem and give methods for placement (at array sites) and embedding (in the mask sequence) of probes in both synchronous and asynchronous regimes. These previous methods do not address several practical details of the application and, more critically, are not scalable to the $O(10^8)$ probes contemplated for next-generation probe arrays. In this work, we make two main contributions:

- We give improved dynamic programming algorithms that perform probe embedding to minimize the number border conflicts while accounting for distance- and position-dependent border conflict weights, as well as the presence of polymorphic probes in the instance.
- We describe and experimentally validate the "engineering" of a scalable, high-quality asynchronous placement heuristic (which is moreover easily parallelizable) for DNA array design. Our heuristic is enabled by a novel approach for simultaneous re-placement and optimal re-embedding of an "independent set" of probes within a small window of the array.

In the process, we draw a number of useful parallels from the 40-year history of placement heuristics in VLSI CAD. Experimental results on both randomly generated and industry testcases confirm that our approach is highly scalable and gives placements of higher quality compared to previous methods.

*Work partially supported by the MARCO Gigascale Silicon Research Center, NSF Grant CCR-9988331, Award No. MM2-3018 of MRDA and CRDF, and by the State of Georgia's Yamacraw Initiative.

RECOMB'03, April 10–13, 2003, Berlin, Germany.

Categories and Subject Descriptors

J.3 [**Life and medical sciences**]: Biology and genetics; J.6 [**Computer-aided engineering**]: Computer-aided design

General Terms

Algorithms, performance, design, experimentation

Keywords

DNA arrays, probe placement, border minimization

1. INTRODUCTION

DNA probe arrays are used in a wide range of genomic analyses, including gene expression monitoring, single nucleotide polymorphism (SNP) mapping, and sequencing by hybridization (see, e.g., [16] for a survey). As described in [8], during very large-scale immobilized polymer synthesis (VLSIPS) the *sites* (cells) 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 transparent regions 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* $S = s_1 s_2 \ldots s_K$ 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. The design of DNA arrays raises a number of combinatorial problems, such as probe selection [15, 18], probe placement [9, 14], manufacturing quality control [1, 12, 19], etc.

Border minimization in DNA array design. We study the *Border Minimization Problem* introduced in [9]

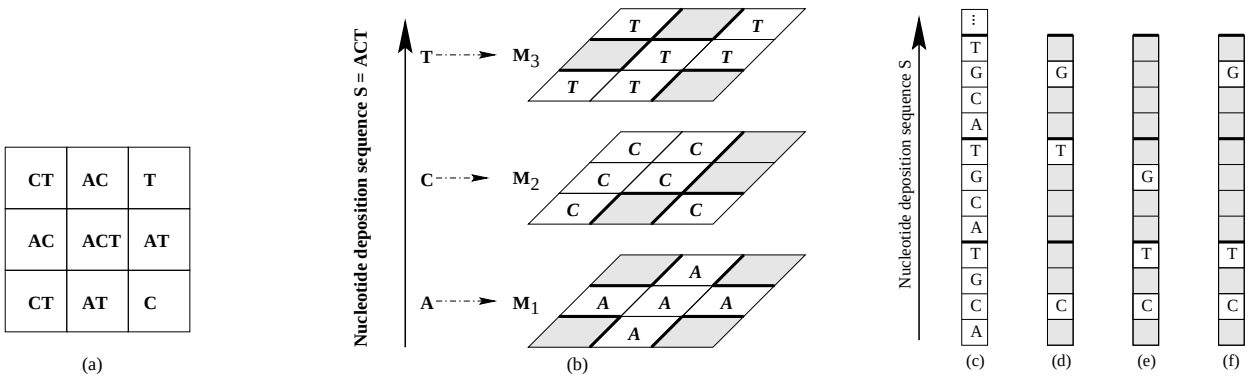

Figure 1: (a) 2-dimensional probe placement. (b) 3-dimensional probe embedding. The nucleotide deposition sequence $S = (ACT)$ corresponds to the sequence of three masks M_1, M_2 and M_3. In each mask the masked sites are shaded and the borders between exposed and masked sites are thickened. (c) Periodic nucleotide deposition sequence S. (d) Synchronous embedding of probe CTG into S; shaded sites denote the masked sites in the corresponding masks. (e-f) Two different asynchronous embeddings of the same probe.

and addressed further in [14]. 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 transparent 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.[1] The *Border Minimization Problem (BMP)* [9, 14] seeks a placement of probes that minimizes the sum of border lengths in all masks.

As in [14], we view array design as a *three-dimensional placement* problem (Figure 1(a-b)): two dimensions represent the site array, and the third dimension represents the sequence S. Each layer in the third dimension corresponds to a mask that induces deposition of a particular nucleotide (A, C, G, or T); a probe is *embedded* within a "column" of 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 columns. The border length of the embedding is the sum of conflict distances between adjacent probes.

We also 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 implies a unique and trivially computed embedding of each probe p in the sequence S; see Figure 1(d). On the other hand, *asynchronous* array synthesis permits arbitrary embeddings, as illustrated in Figure 1(e-f).

Previous work. The work of [14] proposed an *epitaxial growth* method for probe placement which improves over the previous TSP + 1-threading heuristic of [9]. Epitaxial growth places a "seed" probe at the center of the array and adds probes around the seed to greedily minimize the number of induced conflicts. Optimal dynamic programming methods were also proposed which enable re-embedding of one or two adjacent probes to minimize conflicts with already embedded neighbors. Together, these achieve up to 23-30% reductions in border length versus [9]. Finally, lower bounds on optimum synchronous and asynchronous solutions were given.

There are two main motivations for our present work. First, previous methods [9, 14] have at least quadratic time complexity and are hence not scalable. Scalable methods for DNA array design are of great interest, since arrays with up to half a million probes are currently in commercial production, and up to 100 milion probes are envisioned for the near future [16]. Second, previous methods fail to address a number of very important practical details of the application, such as distance- and position-dependent border conflict weights, or the presence of polymorphic probes in the instance [11].

Our contributions. In this work, we develop *high-quality, scalable methods* for designing large DNA arrays. We give improved algorithms for the Border Minimization Problem [9, 14] that can account for distance- and position-dependent border conflict weights, as well as the presence of polymorphic probes (SNPs) in the instance. These methods allow more accurate modeling of the array design problem, and hence form the basis of more practically relevant heuristics. We also describe and experimentally validate the "engineering" of a scalable, high-quality asynchronous placement heuristic for DNA array design. We demonstrate the value of simple ordering-based methods for initial placement. We also propose the use of scalable sliding-window and row-epitaxial techniques having antecedents in large-scale integrated circuit placement [7, 10, 17, 21], as well as a local improvement operator based on 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 scaling of runtime complexity and better solution quality compared to best previous methods.

[1] Reducing unwanted illumination improves the signal to noise ratio in image analysis after hybridization, and thus permits smaller array sites or more probes per array [11].

Organization of the paper. In the next section, we note several aspects of a practical array placement formulation, and describe a dynamic programming algorithm for optimum embedding of polymorphic probes given embeddings of the probes in their local neighborhood. In Section 3, we describe the engineering of a scalable, high-quality placement method. Section 4 gives experimental results on both randomly generated and industry data. We conclude with directions for future research.

2. PROBE EMBEDDING ALGORITHMS

The following extensions to the border length minimization problem are important in practice, but have not been addressed by previous works on the problem [9, 14].

1. **Distance-dependent border conflict weights.** Back-reflection of light affects not only adjacent array cells, but also cells that are as far as 3 cells apart [11]. This implies that we 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 [11]. 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 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 the methods in [14] to handle the first two extensions is straightforward; we focus on the last extension. Define a *probe* to be a set of 1, 2, or 4 k-mers (SNPs) which differ only in the middle position. We first describe embedding of a probe consisting of a single k-mer, and then generalize to the case of two or four SNPs per probe.

In this paper we 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 only lead to a very small loss in solution quality. 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 non-blank 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[2] giving the conflict weight between a masked cell and an unmasked cell as a function of the horizontal and vertical distances between them

[2]The support of a function f is the subset of its domain where f has non-zero values.

Input: Nucleotide deposition sequence $S = s_1 s_2 \dots s_K$, $s_i \in \{A, C, G, T\}$; probe $p = p_1 p_2 \dots p_k$, $p_i \in \{A, C, G, T\}$, 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 $i = 1, \dots, k$ and $j = 1, \dots, K$ using (1) and (2)
2. $cost(0, 0) = 0$; For $i = 1, \dots, k$, $cost(i, 0) = \infty$
3. For $j = 1, \dots, K$ do
   $cost(0, j) = cost(0, j-1) + x_{ij}$
   For $i = 1, \dots, k$ 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

Figure 2: The embedding algorithm for a probe with no mutations

- ω_i, $i = 1, \dots, k$, non-negative 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 (see Figure 2) is essentially computing a shortest path in a directed acyclic graph $G_1 = (V_1, E_1)$ defined as follows (see Figure 3).[3] The vertex set of G_1 is $V_1 = \{0, \dots, k\} \times \{0, \dots, K\}$, where k is the length of p and K is the number of masks, i.e., the length of the nucleotide deposition sequence S. The edge set of G_1 is $E_1 = E_{horiz} \cup E_{diag}$, with

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

and

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

Note that each directed path from $(0, 0)$ to (k, K) consists of K edges, k of which are diagonals. Furthermore, 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 $1 \leq i \leq k$, and the embedding of p corresponding to P has $p_i = s_j$ in the j^{th} position.

Let c_p the array cell assigned to p, and, for every array cell $c \neq c_p$, let q_c be the *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

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

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

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

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

[3]The same underlying graph but with different edge costs is used in [14].

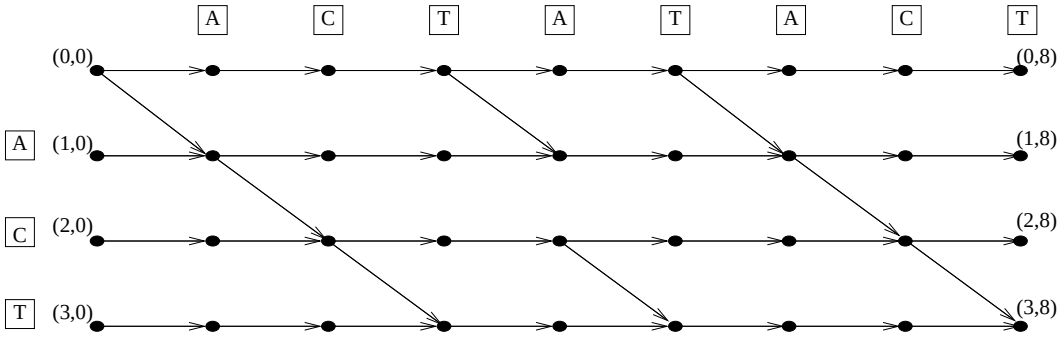

Figure 3: Directed acyclic graph G_1 representing possible embeddings of probe $p = ACT$ into the nucleotide deposition sequence $S = ACTATACT$.

the corresponding embedding of p and the surrounding embedded probes.

THEOREM 1. *The algorithm in Figure 2 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 runtime 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, $p_1 \dots p_{m-1} p_m p_{m+1} \dots p_k$ and $p_1 \dots p_{m-1} p'_m p_{m+1} \dots p_k$. 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 Figure 4) obtained from G_1 by

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

$$\omega(i) \sum \big(w(h(c_p, c), v(c_p, c)) + w(h(c'_p, c), v(c'_p, c))\big) \quad (3)$$

  where the sum is taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j \neq b$
- Changing the cost of each remaining diagonal edge $(i-1, j-1) \to (i, j)$ to

$$\omega(||q_c||_j) \sum \big(w(h(c, c_p), v(c, c_p)) + w(h(c, c'_p), v(c, c'_p))\big) \quad (4)$$

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

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

  where the sums are taken over all $c \notin \{c_p, c'_p\}$ such that $(q_c)_j \neq b$
- 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

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

    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

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

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

$$\begin{aligned} & \omega(m) w(h(c'_p, c_p), v(c'_p, c_p)) \\ + \quad & \omega(||q_c||_j) \sum w(h(c, c_p), v(c, c_p)) \end{aligned} \quad (8)$$

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

$$\begin{aligned} & \omega(m) w(h(c_p, c'_p), v(c_p, c'_p)) \\ + \quad & \omega(||q_c||_j) \sum w(h(c, c'_p), v(c, c'_p)) \end{aligned} \quad (9)$$

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

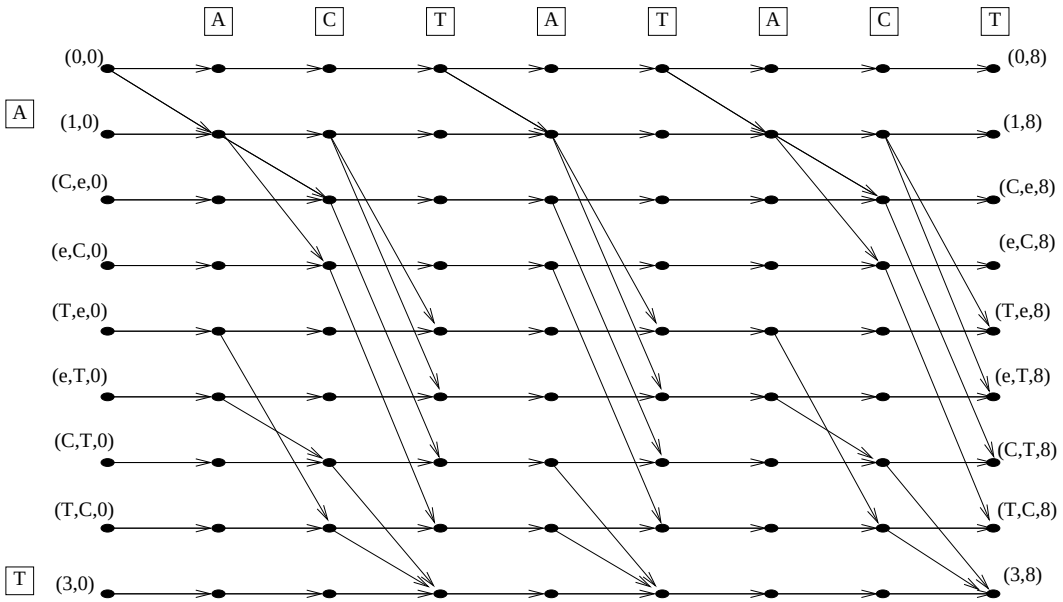

Figure 4: Directed acyclic graph G_2 representing possible embeddings of probe $p = A\{C|T\}T$ into the nucleotide deposition sequence $S = ACTATACT$.

– $(p_m, p'_m, j-1) \rightarrow (m+1, j)$ 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 Figure 2.

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:

THEOREM 2. *The minimum conflict embedding of a two or four SNP probe can be computed in $O(kK + KW)$ time.*

3. ENGINEERING OF A SCALABLE, HIGH-QUALITY PLACEMENT HEURISTIC

In this section, we describe the "engineering" of near linear-time, yet high-quality 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.

We begin by noting two design goals: (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 next note that the placer design can be made according to two basic functions [21]: *initial placement* and (iterative) *placement improvement*. Each of these functions can apply to (i) the assignment of probes to array sites, and/or (ii) the embedding of probes in the mask sequence S. Within an overall placement metaheuristic, the requirements for one function are strongly dependent on the implementation of the other function.

3.1 Scalable Initial Placement

The method of [9] uses multi-fragment greedy TSP and "1-threading," along with synchronous embedding (allowing Hamming distance between probes to capture border conflict cost), to place probes into array sites. Since the TSP heuristic is based on a complete distance matrix, its time complexity is quadratic in the number of probes. The method of [14] is based on epitaxial growth, which also requires quadratic time complexity since every unplaced probe is a candidate for the site currently being filled. The constant factor in the asymptotic analysis is quite large, since border conflict costs are computed by dynamic programming (asynchronous embedding) rather than by Hamming distance (synchronous embedding).

- *Observation 1: Existing methods can be "trivially scaled."* "Trivial scaling" partitions the set of probes and the probe array each into K subsets ("chunks"), then solves for K independent initial placements using a given previous method. While runtime remains linear in the number of probes, two types of losses are incurred: (i) from lack of freedom of a probe to move anywhere other than its subset's assigned chunk of array sites, and (ii) lack of optimization on borders between chunks.
- *Observation 2: Linear-time tours (orderings) are possible.* Bounded probe length (e.g., 25-mers) and the finite alphabet from which probes are formed enable fast, linear-time analogs of the TSP tour used by [9]. For example, the spacefilling curve (Gray code) ordering of

[3] may be applied, and indeed has had success in the VLSI context [2]. An even simpler ordering is found by linear-time radix sort; after this "tour" has been found, 1-threading and succeeding steps may follow as in [9, 14].

- *Observation 3: Probe array instances have no natural topology or locality.* Unlike VLSI placement, where the placeable objects (Boolean gates) are connected by a sparse "netlist" of logic signals, DNA placement instances have no natural topology or locality in the probe sets. Moreover, computing a useful topology (e.g., from clustering or other analyses) is difficult to perform in near-linear time. Thus, a large number of VLSI placement methods such as force-directed, spectral (using eigenvectors of the Laplacian of a weighted adjacency matrix), etc. - as well as their well-scaling multilevel variants [5] - are not obviously applicable to DNA arrays.[4]

Based on Observation 1, we have implemented trivial scaling with the best previous method of [14] and chunk sizes up to 50x50. However, the results are worse than those of other methods proposed below, and we do not consider trivial scaling further. Based on Observation 2, we have implemented a simple initial placement based on radix sort followed by the 1-threading of [9].[5]

3.2 Scalable 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 [7] iteratively determines an optimal reassignment of all objects within a given "window" of the placement. The "end-case" placer of [4] uses branch and bound to optimally reorder small sub-rows 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" [7] around the placement; cf. "cycling and overlapping" [10], "row-ironing" [4], 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,[6] we focus on "strong" operators.

- *Observation 4: Optimal probe re-embedding by itself is scalable, but ineffective.* Using the probe alignment (i.e., embedding) algorithms in Section 2 as subroutines, the post-placement optimization algorithms of [14] can be modified to address the practical extensions (weighting, polymorphism, etc.) noted at the beginning of Section 2. In particular, dynamic programming methods for optimal alignment enable us to apply the linearly scaling "chessboard" algorithm of [14].[7] However, chessboard re-embedding is weak, achieving only 0.02% reduction in border length when applied after other methods that we describe.
- *Observation 5: Simultaneous probe re-placement and re-embedding of an entire "window" is not practical.* Because the optimal alignment of a probe depends on the alignments of its neighbors, there is no practical way to simultaneously re-place and re-embed, say, six probes that are placed in a 2x3 array of sites. Thus, analogs of the DOMINO [7] and end-case placer [4] VLSI placement approaches are, to our knowledge, infeasible.
- *Observation 6: Simultaneous probe re-placement and re-embedding of an "independent set" within a window is practical.* Driven by Observation 6, 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 "independent" array cells (i.e., a set for which every two cells c, c' satisfy $w(h(c,c'), v(c,c')) = w(h(c',c), v(c',c)) = 0$), then reassign 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 minimum embedding cost computed using the algorithms in Section 2. For a set of t independent cells, computing all minimum-cost embeddings requires $O(t^2(kK+KW))$ time, while computing the minimum cost assignment requires an additional $O(t^3)$ time. As noted above, full-chip application with practical runtime is achieved by iteratively choosing the independent set from a "sliding window" that is moved around the array.[8] The matching approach is a reminiscence of the early work on electronic circuit placement [20, 13].

3.3 The Sliding-Window Matching Algorithm

We have implemented the "sliding window" method described above, as follows. (1) We first radix-sort all probes into the obvious lexicographic order and then perform 1-threading as in [9]. (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

[4] Scalable clustering methods have been developed for analysis of large literatures or information spaces (e.g., [6]); we have not yet examined their relevance to clustering and eventual placement of large sets of probes.

[5] In the experimental discussion, we refer to this as "sorting + 1-threading." We sort probes lexicographically by position 1, position 2, etc. Quality can be improved by using a space-filling curve embedding and by ordering lexicographically with respect to the positions that have highest conflict weights, but we do not discuss such refinements below.

[6] And the intuition that randomly chosen pairs of optimally-embedded probes are extremely unlikely to be swappable with reduction in border cost.

[7] For a fixed coloring of the graph in which two probes are adjacent iff they border one another, the chessboard algorithm iteratively aligns all probes in each color class with respect to their neighbors.

[8] 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 find that use of a single random maximal independent set ($K = 1$) yields the best solution quality vs. runtime tradeoff point.

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.[9]

Figure 5 illustrates the heuristic tuning with respect to window sizes and window overlaps. We observe that (i) larger window sizes and fewer passes lose out to smaller window sizes and more passes; and (ii) overlap size = half the window size is a good tradeoff point. Thus, 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 (iii) 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. Finally, we examined interactions between effort level in initial placement and in placement improvement. Among the more solid observations is that (iv) it is not cost-effective to optimize embedding (i.e., probe alignments) during initial placement. Thus, we pass a synchronously embedded probe placement to the placement improvement phase.

To further speed up placement we have also devised a "synchronous variant" of Sliding-Window Matching that uses a modified Step (2'): for each sliding $W_0 \times W_0$ window we choose one random maximal independent set of sites and determine the cost of *synchronous* reassignment of each associated probe to each site, then reassign probes according to the minimum weight perfect matching in the resulting weighted bipartite graph. To compensate for the worsened solution quality, we add a postprocessing Step (5): run chessboard postplacement [14] on the synchronous placement that results after window-sliding terminates. Last, we emphasize that both the Sliding-Window Matching and its Synchronous Variant are easily parallelizable after the initial (linear-time) sorting and 1-threading step. Previous methods [9, 14] do not have this property.

3.4 The Row-Epitaxial Algorithm

Epitaxial, or "seeded crystal growth," placement is a technique that has been well-explored in the VLSI circuit placement literature [17, 21]. The technique essentially grows a 2-dimensional placement around a single starting "seed." Intuitively, 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. 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 [14]) 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. In fact, a pair of a site and a probe filling this site is chosen simultaneously. Although this algorithm achieves better results comparatively with other 2-dimensional placements, it becomes impractical for DNA chips with dimensions more than 300 × 300.

[9]Faster variants can restrict the number of such passes to a constant, e.g., 5. Typically, our implementation makes around 20 passes.

In this paper we suggest a scalable variant of the original version, so called the *row-epitaxial* algorithm. There are three main distinguished features of the row-epitaxial variant:

(1) It reshuffles an already existing pre-optimized placement rather than starting with an empty placement;

(2) The sites are filled with crystalized probes in a fixed specified order, namely, row by row from left to right, so that on average each next site is adjacent to two already "crystalized" neighbors;

(3) The probe for filling the next site is chosen among limited amount of not yet "crystalized" probes that lie in close proximity, i.e., next k_0 rows, to this site.

Feature (1) allows to greatly improve the quality of the row-epitaxial algorithm. The choice of the initial pre-optimized placement is very limited, e.g., any 2-dimensional placement based on computing pairwise distances between probes such as TSP-based placement (see [9]) becomes impractical for DNA chips of size more than 500 × 500. Instead we found that an excellent substitution of TSP is lexicographical probe sorting.

Feature (2) reduces the runtime of row-epitaxial algorithm by factor of $O(n^2)$ for an $n \times n$ array, since instead of the best (site,probe) pair, it finds only the best probe for a certain fixed site. The further reduction in runtime implied by feature (3) is very flexible: the number k_0 of the "look-ahead" rows can be adjusted with respect to the array size n.

4. EXPERIMENTAL RESULTS

We have implemented the Sliding-Window Matching algorithm of Section 3.3 and its synchronous variant. For comparison we also include results for the TSP+1-Threading and Synchronous Epitaxial placements, followed by chessboard postplacement alignment, from [14]. All algorithms were coded in C and compiled using `g++` version 2.95.3 with `-O3` optimization. Placement algorithms were run on a dual-processor 1.4GHz Intel Xeon server with 512MB RAM. Timing was performed under similar load conditions for all experiments.

Table 1 gives the results of experiments performed on arrays of random 25-mer probes, with array size ranging from 20×20 to 1000×1000. As in [14], in these experiments there are no SNPs, we do not consider conflicts between non-adjacent probes, and we give equal conflict weight to all positions in a probe. All reported numbers are averages over 10 different randomly generated arrays of the given size.

The results indicate that our methods are very scalable and compare very favorably to best previous results (epitaxial placement + chessboard alignment from [14]) in terms of solution quality. At instance sizes of 500x500 probes, the row-epitaxial algorithm and the asynchronous sliding-window matching reduce border cost by 9%, respectively 7%, versus [9] while requiring a fraction of the runtime. The synchronous variant is within a few percents of row-epitaxial and asynchronous SWM, and requires only 1300 seconds for million-probe arrays which previous methods cannot handle. Parallelized execution can achieve further speedups and/or permit the use of stronger local improvement operators, e.g., we estimate that increasing sliding window size to 12x12 will improve solution quality by a further 5%.

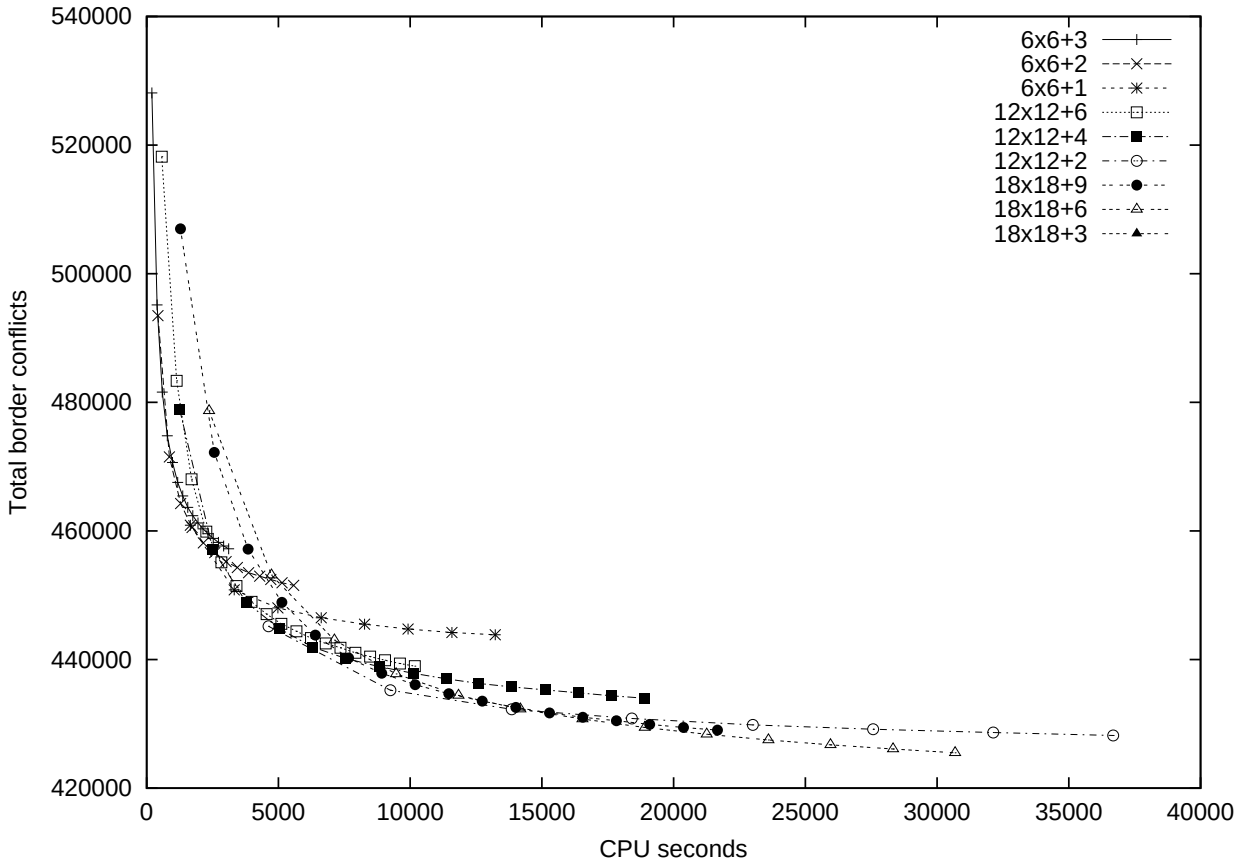

Figure 5: Solution quality vs. runtime for Synchronous Sliding-Window Matching with varying window size and window overlap; array size = 100×100.

We have also run 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 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 rightmost embedding; (2) Lexicographical sorting of the embedded probes followed by 1-threading; (3) Synchronous sliding window matching – we used 48×48 windows with overlap 24 – where "synchronous" here refers to computing Hamming distances between embedded probes rather than un-embedded probes; (4) Row-epitaxial with $k_0 = 80$ rows of lookahead; and (5) A SNP-aware version of the Chessboard alignment algorithm. The final number of conflicts was reduced by 4.2% with respect to the Affymetrix optimized placement.[10]

[10]We understand [11] that Affymetrix uses placement techniques that are similar to row-epitaxial placement combined with rightmost alignment. This probably explains why the 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.

5. CONCLUSIONS

In this paper we have (1) extended dynamic programming-based optimal alignment to address practical criteria (distance- and position-dependent conflict weights, presence of polymorphism) and (2) described a highly-scalable heuristic for asynchronous probe array placement. Our placement heuristic gives improved solution quality while achieving large speedups versus previous methods; it also gives opportunities to trade CPU time and parallel implementation for solution quality. Our approach compares favorably to commercial array placements. We are currently optimizing our implementation for multiprocessing and practical cost criteria. Here, other metaheuristic frameworks (e.g., large-step Markov chains) are of interest, as well as 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.

Acknowledgments. We thank Earl Hubbell and Michael Mittman for providing us many relevant details of the problem and the Human Genome chip data used in the experimental study. We also thank the anonymous referees for several suggestions that improved the presentation.

6. REFERENCES

[1] N. Alon, C. J. Colbourn, A. C. H. Lingi and M. Tompa, "Equireplicate Balanced Binary Codes for

Chip Size	Random Placement	TSP+1Thr+Chess.		Epit.+Chess.		Row-Epit.+Chess.		SyncSWM+Chess.		AsyncSWM	
		Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU	Cost	CPU
20	28509	17704	0	17250	0	17186	2	17726	0	17068	17
40	116908	71426	3	68847	6	68260	14	70782	0	68290	133
60	265477	160013	15	153444	32	152332	36	157868	0	151736	338
80	473981	282848	46	270211	104	266930	73	279504	0	269196	576
100	742331	439829	113	419069	274	413158	118	433274	1	417890	875
200	2984879	1723352	1901	1624988	4441	1593146	493	1693658	46	1636658	3676
300	6727176	3801765	12028	—	—	3503526	1562	3746722	112	3615282	8406
500	18712407	10426237	109648	—	—	9418042	8400	10049442	302	9686918	22351
1000	74924959	—	—	—	—	35918568	41740	38898792	1307	—	—

Table 1: Total conflicts and CPU time of the compared heuristics (averages over 10 random instances). The AsyncSWM entry for 1000x1000 was stopped after 10 iterations. We use 6×6 windows with overlap 3 for both synchronous and asynchronous SWM, and $k_0 = 20$ look-ahead rows for Row-Epitaxial.

Oligo Arrays," *SIAM Journal on Discrete Mathematics* 14(4) (2001), pp. 481-497.

[2] C. J. Alpert and A. B. Kahng, "Multi-Way Partitioning Via Spacefilling Curves and Dynamic Programming," *Proc. ACM/IEEE Design Automation Conf.*, 1994, pp. 652-657.

[3] J. J. Bartholdi and L. K. Platzman, "An O(N log N) Planar Travelling Salesman Heuristic Based On Spacefilling Curves," *Operations Research Letters* 1(4) (1982), pp. 121-125.

[4] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-Case Placers for Standard-Cell Layout," *Proc. ACM Intl. Symp. on Physical Design*, 1999, pp. 90-96.

[5] T. F. Chan, J. Cong, T. Kong and J. R. Shinnerl, "Multilevel Optimization for Large-Scale Circuit Placement," *Proc. IEEE/ACM Intl. Conf. on Computer Aided Design*, 2000, pp. 171-176.

[6] D. R. Cutting, D. R. Karger, J. O. Pederson and J. W. Tukey, "Scatter/Gather: A Cluster-Based Approach to Browsing Large Document Collections," (15th Intl. ACM/SIGIR Conference on Research and Development in Information Retrieval) *SIGIR Forum* (1992), pp. 318-329.

[7] K. Doll, F. M. Johannes, K. J. Antreich, "Iterative Placement Improvement by Network Flow Methods," *IEEE Transactions on Computer-Aided Design* 13(10) (1994), pp. 1189-1200.

[8] 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.

[9] 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.

[10] D. J. Huang and A. B. Kahng, "Partitioning-Based Standard Cell Global Placement With an Exact Objective," *Proc. ACM Intl. Symp. on Physical Design*, 1997, pp. 18-25.

[11] E. Hubbell and M. Mittmann, *personal communication* (Affymetrix, Santa Clara, CA), July 2002.

[12] 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.

[13] S. Akers, "On the Use of the Linear Assignment Algorithm in Module Placement," *Proc. 18th Design Automation Conference*, 1981, pp. 137–144. pp. 113–117.

[14] A.B. Kahng, I.I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky, "Border Length Minimization in DNA Array Design," *Proc. 2nd International Workshop on Algorithms in Bioinformatics (WABI 2002),* Springer-Verlag Lecture Notes in Computer Science Series, to appear.

[15] 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.

[16] R.J. Lipshutz, S.P. Fodor, T.R. Gingeras, D.J. Lockhart, "High density synthetic oligonucleotide arrays," *Nat. Genet.* 21 (1999), pp. 20–24.

[17] B. T. Preas and M. J. Lorenzetti, eds., *Physical Design Automation of VLSI Systems*, Benjamin-Cummings, 1988.

[18] S. Rahmann. "Rapid large-scale oligonucleotide selection for microarrays," *Proc. IEEE Computer Society Bioinformatics Conference (CSB)*, 2002.

[19] R. Sengupta and M. Tompa, "Quality Control in Manufacturing Oligo Arrays: a Combinatorial Design Approach," *Journal of Computational Biology* 9 (2002), pp. 1–22.

[20] L. Steinberg, "The Backboard Wiring Problem: A Placement Algorithm," *SIAM Review* 3(1) (1961), pp. 37–50.

[21] K. Shahookar and P. Mazumder, "VLSI Cell Placement Techniques," *Computing Surveys* 23(2) (1991), pp. 143-220.