Computer-Aided Optimization of DNA Array Design and Manufacturing

Andrew B. Kahng, *Member, IEEE*, Ion I. Măndoiu, Sherief Reda, *Student Member, IEEE*, Xu Xu, and Alexander Z. Zelikovsky

***Abstract*—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. DNA chips are manufactured through a highly scalable process called very large-scale immobilized polymer synthesis (VLSIPS) that 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 100 million probes in the next generation. This paper is one of the first attempts to apply very large scale integration (VLSI) computer-aided design methods to the physical design of DNA chips, where the main objective is to minimize total border cost (i.e., the number of nucleotide mismatches between adjacent sites).**

By exploiting analogies between manufacturing processes for DNA arrays and for VLSI chips, the authors demonstrate the potential for transfer of methodologies from the 40-year-old field of electronic design automation to the newer DNA array design field. The main contributions of this paper are the following. First, it proposes several partitioning-based algorithms for DNA probe placement that improve solution quality by over 4% compared to best previously known methods. Second, it gives a new design flow for DNA arrays, which enhances current methodologies by adding flow awareness to each optimization step and introducing feedback loops. Third, it proposes solution methods for new formulations integrating multiple design steps, including probe selection, placement, and embedding. Finally, it introduces new techniques to experimentally evaluate the scalability and suboptimality of existing and newly proposed probe placement algorithms. Interestingly, the authors find that DNA placement algorithms appear to have better suboptimality properties than those recently reported for VLSI placement algorithms (Chang *et al.*, 2003 and Cong *et al.*, 2003).

Manuscript received January 10, 2005; revised June 20, 2005. This work was supported in part by Cadence Design Systems, Inc., in part by the MARCO Gigascale Silicon Research Center, in part by the National Institutes of Health (NIH) under Award 1 P20 GM065762-01A1, and in part by the University of Connecticut's Research Foundation. Preliminary versions of the results in this paper have appeared in the Proceedings of the International Conference on Computer Design, San Jose, CA, 2003 and the Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, 2003. This paper was recommended by Associate Editor K. Chakrabarty.

A. B. Kahng is with the Department of Computer Science and Engineering and Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: abk@ucsd.edu).

I. I. Măndoiu was with the Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA. He is now with the Computer Science and Engineering Department, University of Connecticut, Storrs, CT 06269-2155 USA (e-mail: ion@engr.uconn.edu).

S. Reda and X. Xu are with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093-0114 USA (e-mail: sreda@cs.ucsd.edu; xuxu@cs.ucsd.edu).

A. Z. Zelikovsky is with the Computer Science Department, Georgia State University, Atlanta, GA 30303 USA (e-mail: alexz@cs.gsu.edu).

Digital Object Identifier 10.1109/TCAD.2005.855940

***Index Terms*—DNA array design flow, flow-aware optimizations, probe placement and embedding, scaling suboptimality.**

I. Introduction

DNA PROBE arrays—DNA arrays or DNA chips for short—have recently emerged as one of the core genome technologies. They provide a cost-effective method for obtaining fast and accurate results in a wide range of genomic analyses, including gene expression monitoring, mutation detection, and single nucleotide polymorphism analysis (see [41] for a survey). The number of applications is growing at an exponential rate [25], [52], already covering a diversity of fields ranging from health care to environmental sciences and law enforcement. The reasons for this rapid acceptance of DNA arrays are a unique combination of robust manufacturing, massive parallel measurement capabilities, and highly accurate and reproducible results.

Today, most DNA arrays are manufactured through a highly scalable process, referred to as very large-scale immobilized polymer synthesis (VLSIPS), that combines photolithographic technologies adapted from the semiconductor industry with combinatorial chemistry [1], [2], [22]. Similar to very large scale integration (VLSI) circuit manufacturing, multiple copies of a DNA array are simultaneously synthesized on a wafer, typically made out of quartz. To initiate synthesis, linker molecules including a photo-labile protective group are attached to the wafer, forming a regular two-dimensional (2-D) pattern of synthesis sites. Probe synthesis then proceeds in successive steps, with one nucleotide $(A, C, T, \text{or } G)$ being synthesized at a selected set of sites in each step. To select which sites will receive nucleotides, photolithographic masks are placed over the wafer. Exposure to light de-protects linker molecules at the nonmasked sites. Once the desired sites have been activated in this way, a solution containing a single type of nucleotide (which bears its own photo-labile protection group to prevent the probe from growing by more than one nucleotide) is flushed over the wafer's surface. Protected nucleotides attach to the unprotected linkers, initiating the probe synthesis process. In each subsequent step, a new mask is used to enable selective de-protection and single-nucleotide synthesis. This cycle is repeated until all probes have been fully synthesized.

As the number of DNA array designs is expected to ramp up in the coming years with the ever-growing number of applications [25], [52], there is an urgent need for high-quality software tools to assist in the design and manufacturing process. The biggest challenges to the rapid growth of DNA array

technology are the drastic increase in design sizes with simultaneous decrease of array cell sizes—next-generation designs are envisioned to have hundreds of millions of cells of submicron size [2], [41]—and the increased complexity of the design process, which leads to unpredictability of design quality and design turnaround time. Surprisingly enough, despite huge research efforts invested in DNA array applications, very few works are devoted to computer-aided optimization of DNA array design and manufacturing. Current design practices are dominated by *ad hoc* heuristics incorporated in proprietary tools with unknown suboptimality. This will soon become a bottleneck for the next generation of high-density arrays, such as the ones currently being designed at Perlegen [2].

In this paper, we exploit the similarities between manufacturing processes for DNA arrays and VLSI chips and demonstrate the significant potential for transfer of electronic design automation methodologies [19], [48] to the newer DNA array design field. Our main contributions in this paper are the following.

- A new DNA probe array placement algorithm that recursively places the probes on the chip in a manner similar to top-down VLSI placers, via a centroid-based strategy, and a new technique for asynchronous re-embedding of placed probes within the mask sequence. Experimental results show that combining the new algorithms results in an average improvement of 4.0% over best previous flows (Section III).
- A new design flow for DNA arrays, which enhances current methodologies by adding flow awareness to each optimization step and introducing feedback loops (Section IV). In particular, we propose new solution methods that integrate probe placement and embedding with probe selection (Section IV-A).
- A comprehensive experimental study demonstrating significant solution quality improvements for the enhanced methodologies. In particular, we show that 5%–7% improvement in border length can be achieved over the highest quality scalable flow previously reported in the literature [34], [37] by a tighter integration of probe placement and embedding (Section IV-B). Furthermore, we show that an additional improvement in border length of up to 15% can be achieved by integrating probe selection with probe placement and embedding (Section IV-C).
- New techniques for studying and quantifying the performance of probe placement and embedding algorithms, including the development of benchmarks with known optimal cost and scaling suboptimality experiments similar to recent studies in the VLSI computer-aided design (CAD) field (Section V).

This paper is organized as follows. Section II introduces the various steps in the DNA array design flow and the problems addressed in this work. Section III briefly summarizes previous work on DNA array physical design. Section IV gives the new partitioning-based probe placement algorithm. We also analyze the proposed algorithm runtime complexity and compare its performance against other border minimization algorithms. Section V presents new enhancements for the DNA array design flow. These enhancements, inspired by similar techniques developed in VLSI CAD design, lead to further reductions in border length. Finally, Section VI quantifies the suboptimality and optimality of various probe placement and embedding heuristics.

II. DNA Array Design Flow

In this section, we introduce the main steps of the design flow for DNA arrays, noting the similarity to the VLSI design flow and briefly reviewing previous work. The application of this flow to the design of a DNA chip for studying gene expression in the Herpes B virus is described in [8]. We later discuss (in Section IV) how the current DNA array design flow may be enhanced by adding flow awareness to each optimization step and introducing feedback loops between steps—techniques that have proved very effective in the VLSI design context [19], [48].

A. Probe Selection

Analogous to logic synthesis in VLSI design, the probe selection step is responsible for implementing the desired functionality of the DNA array. Although probe selection is application dependent, several underlying selection criteria are common to all designs, regardless of the intended application [1], [2], [7], [32], [40], [43].

First, in order to meet array functionality, the selected probes must have low hybridization energy for their intended targets and high hybridization energy for all other target sequences. Hence, a standard way of selecting probes is to select a probe of minimum hybridization energy from the set of probes that maximizes the minimum number of mismatches with all other sequences [40]. Second, since selected probes must hybridize under similar operating conditions, they must have similar melting temperatures.[1] Finally, to simplify array design, probes are often constrained to be substrings of a predetermined nucleotide deposition sequence. Typically, there are multiple probe candidates satisfying these constraints.

B. Deposition Sequence Design

The number of synthesis steps directly affects manufacturing time and the number of masks in the mask set, and also directly affects the quantity of defective probes synthesized on the chip. Therefore, a basic optimization in DNA array design is to minimize the number of synthesis steps. In the simplest model, this optimization has been reformulated as the classical shortest common supersequence (SCS) problem [38], [50]: given a finite alphabet Σ (for DNA arrays $\Sigma = \{A, C, T, G\}$) and a set $P = \{p_1, \ldots, p_t\} \subseteq \Sigma^n$ of probes, find a minimum-length string $s_{\text{opt}} \in \Sigma^*$ such that every string of P is a subsequence of s_{opt}.

[1] At the melting temperature, two complementary strands of DNA are as likely to be bound to each other as they are to be separated. A practical method for estimating the melting temperature is suggested in [32].

(A string p_i is a subsequence of s_{opt} if s_{opt} can be obtained from p_i by inserting zero or more symbols from Σ.) The SCS problem has been studied for over two decades from the point of view of computational complexity, probabilistic and worst-case analysis, approximation algorithms and heuristics, experimental studies, etc. (see, e.g., [9]–[11], [17], [23], [24], [31], and [44]).

The general SCS problem is NP-hard and cannot be approximated within a constant factor in polynomial time unless $P = NP$ [31]. On the other hand, a $|\Sigma|$ approximation is produced by using the trivial periodic supersequence $s = (x_1, x_2, \ldots, x_{|\Sigma|})^n$, where $\Sigma = \{x_1, x_2, \ldots, x_{|\Sigma|}\}$. Better results are produced in practice by a simple greedy algorithm usually referred to as the "majority merge" algorithm [23], or variations of it that add randomization, lookahead, bidirectionality, etc. (see, e.g., [38]).

Current DNA array design methodologies bypass the deposition design step and use a predefined typically periodic deposition sequence such as $ACTGACTG\ldots$ (see, e.g., [38], [50]).

C. Design of Control and Test Structures

DNA array manufacturing defects can be classified as noncatastrophic, i.e., defects that affect the reliability of hybridization results but do not compromise chip functionality when maintained within reasonable limits, and catastrophic, i.e., defects that render the chip unusable. Noncatastrophic defects are caused by systematic error sources in the VLSIPS manufacturing process, such as unintended illumination due to diffraction, internal reflection, and scattering. Their impact on the hybridization reliability of the chip is reduced by using the Perfect Match/Mismatch strategy [1], [41]. Under this strategy, a so-called "mismatch probe" is synthesized next to each functional probe ("perfect match probe"). The sequence of the mismatch probe is identical to that of the perfect match probe, except for the middle nucleotide, which is replaced with its Watson–Crick complement. To reduce the effect of noncatastrophic manufacturing defects and of nonspecific hybridization, under the standard data analysis protocol the hybridization signal is obtained by subtracting the fluorescence intensity of the mismatch probe from that of the perfect match probe.

Catastrophic manufacturing defects affect a large fraction of the probes on the chip and are typically caused by missing, out-of-order, or incomplete synthesis steps, wrong or misaligned masks, etc. These defects can be detected using test structures similar to built-in self-test (BIST) structures in VLSI design. A common approach is to synthesize a small set of test probes (sometimes referred to as fidelity probes [29]) on the chip and add their fluorescently labeled complements to the genomic sample that is hybridized to the chip. Multiple copies of each fidelity probe are deliberately manufactured at different locations on the chip using different sequences of synthesis steps. Lack of hybridization at some of the locations where fidelity probes are synthesized can be used not only to detect catastrophic manufacturing defects but also to identify the erroneous manufacturing steps. Further results on test structure design for DNA chips include those in [6], [14], and [46].

D. Physical Design

The physical design for DNA arrays is equivalent to the physical design phase in VLSI design. It consists of two steps: probe placement, which is responsible for mapping selected probes onto locations on the chip, and probe embedding, which embeds each probe into the deposition sequence (i.e., determines synthesis steps for all nucleotides in the probe). The result of probe placement and embedding is the complete description of the reticles used to manufacture the array.

Under ideal manufacturing conditions, the functionality of a DNA array is not affected by the placement of the probes on the chip or by the probe synthesis schedules. In practice, since the manufacturing process is prone to errors, probe locations and synthesis schedules affect to a great degree the hybridization sensitivity and ultimately the functionality of the DNA array. There are several types of synthesis errors that take place during array manufacturing. First, a probe may not loose its protective group when exposed to light, or the protective group may be lost but the nucleotide to be synthesized may not attach to the probe. Second, due to diffraction, internal reflection, and scattering, unintended illumination may occur at sites that are geometrically close to intentionally exposed regions. The first type of manufacturing errors can be effectively controlled by careful choice of manufacturing process parameters, e.g., by proper control of exposure times and by insertion of correction steps that irrevocably end synthesis of all probes that are unprotected at the end of a synthesis step [1]. Errors of the second type result in synthesis of unforeseen sequences in masked sites and can compromise interpretation of hybridization intensities. To reduce such uncertainty, one can exploit the freedom available in assigning probes to array sites during placement and in choosing among multiple probe embeddings, when available. The objective of probe placement and embedding algorithms is therefore to minimize the sum of border lengths in all masks, which directly corresponds to the magnitude of the unintended illumination effects.[2] Reducing these effects improves the signal to noise ratio in image analysis after hybridization and thus permits smaller array sites or more probes per array [30].[3]

Let $M_1, M_2, \ldots, M_K$ denote the sequence of masks used in the synthesis of an array, and let $e_i \in \{A, C, T, G\}$ be the nucleotide synthesized after exposing mask M_i. Every probe in the array must be a subsequence of the nucleotide deposition sequence $S = e_1, e_2, \ldots, e_K$. In case a probe corresponds to multiple subsequences of S, one such subsequence, or "embedding" of the probe into S, must be chosen as the synthesis schedule for the probe. Clearly, the geometry of the masks is uniquely determined by the placement of the probes on the array and the particular synthesis schedule used for each probe.

[2]Compared to VLSI physical design, where multiple design metrics (including area, wire length, timing, power consumption, etc.) must be optimized simultaneously, DNA array physical design is simpler in that it must optimize a single objective, namely, total border length.

[3]Unfortunately, the lack of publicly available information about DNA array manufacturing yield makes it impossible to assign a concrete economic value to decreases in total border length.

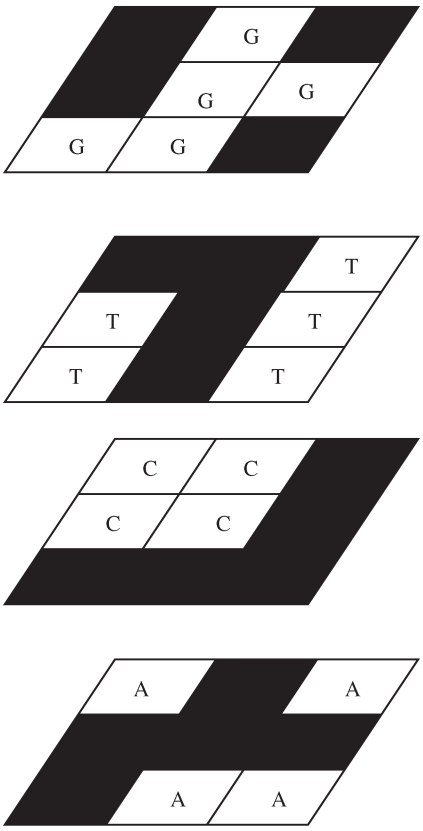


Fig. 1. Three-dimensional probe placement with four masks and $S = ACTG$. Total border length is 24 (seven on the A mask, four on the C mask, six on the T mask, and seven on the G mask).

Formally, the border minimization problem (BMP) is equivalent to finding a three-dimensional (3-D) placement of the probes [33]: two dimensions represent the site array, and the third dimension represents the nucleotide deposition sequence S (see Fig. 1). 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 3-D 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 total border length of a 3-D placement is the sum of conflict distances between adjacent probes, and the BMP seeks to minimize this quantity.

A special case is that of a synchronous synthesis regime, in which the nucleotide deposition sequence S is periodic, and the kth period $(ACGT)$ of S is used to synthesize a single (the kth) additional nucleotide in each probe. Since in this case the embedding of a probe is predefined, the problem reduces to finding a 2-D placement of the probes. The border length contribution from two probes p and p' placed next to each other (in the synchronous synthesis regime) is simply twice the Hamming distance between them, i.e., twice the number of positions in which they differ.

Previous Work on Border Length Minimization: The BMP was first considered for uniform arrays (i.e., arrays containing all possible probes of a given length) by Feldman and Pevzner [20], who proposed an optimal solution based on 2-D Gray codes. Hannenhalli *et al.* [26] gave heuristics for the special case of synchronous synthesis. Their method is to order the probes in a traveling salesman problem (TSP) tour that heuristically minimizes the total Hamming distance between neighboring probes. The tour is then threaded into the 2-D array of sites using a technique similar to the one previously used in VLSI design [39]. For the same synchronous context, Kahng *et al.* [33] suggested an epitaxial, or "seeded crystal growth," placement heuristic similar to heuristics explored in the VLSI circuit placement literature in [42] and [47]. Very recently, Kahng *et al.* [34], [37] proposed methods with near-linear runtime combining simple ordering-based heuristics for initial placement, such as lexicographic sorting followed by threading, with heuristics for placement improvement, such optimal reassignment of an "independent" set of probes [49] chosen from a sliding window [18], or a row-based implementation of the epitaxial algorithm that speeds-up the computation by considering only a limited number of candidates when filling each array site.[4] Previous approaches can be summarized as follows.

1) *TSP + Threading [26]:* This algorithm computes a TSP tour in the complete graph with the probes as vertices and edge costs given by pair-wise Hamming distances. The tour is then threaded into the 2-D array of sites using the 1-threading method described in [26].
2) *Row-Epitaxial [34], [37]:* An implementation of the epitaxial algorithm in [33], where the computation is sped up by a) filling array sites in a predefined order (row by row), and b) considering only a limited number of candidate probes when filling each array site. Unless otherwise specified, the number of candidates is bounded by 20 000 in our experiments.
3) *Sliding-Window Matching (SWM) [34], [37]:* After an initial placement is obtained by 1-threading of the probes in lexicographic order, this algorithm iteratively improves the placement by selecting an "independent" set of probes from a sliding window and then optimally replacing them using a minimum-weight perfect matching algorithm (cf. "row ironing" [12]).

The general BMP, which allows arbitrary or asynchronous probe embeddings [see Fig. 2(c)], was introduced by Kahng *et al.* [33]. They proposed a dynamic programming algorithm that embeds a given probe optimally with respect to fixed embeddings of the probe's neighbors. This algorithm is used as a building block for designing several algorithms that improve a placement by re-embedding probes but without replacing them. An important aspect of probe re-embedding is the probe processing order of re-embedding, i.e., the order that specifies when a probe gets re-embedded. Each of the following two algorithms uses the dynamic programming algorithm in [33] for optimal re-embedding of a probe with respect to the

[4]The work in [34] and [37] also extends probe placement algorithms to handle practical concerns such as preplaced control probes, presence of polymorphic probes, unintended illumination between nonadjacent array sites, and position-dependent border conflict weights.

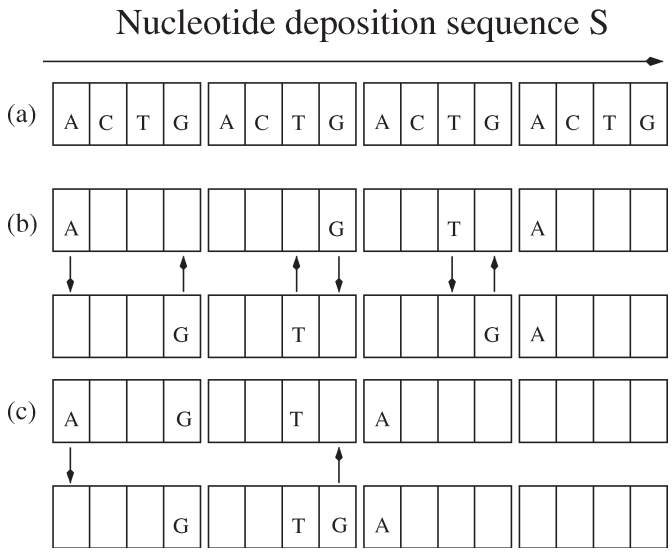


Fig. 2. (a) Periodic deposition sequence. (b) Synchronous embedding of the probes $AGTA$ and $GTGA$ gives six border conflicts (indicated by arrows). (c) "As soon as possible" asynchronous embedding of the probes $AGTA$ and $GTGA$ gives only two border conflicts.

embeddings of its neighbors. They only differ in the processing order of probe re-embedding.

1) *Batched greedy [33]:* This algorithm optimally re-embeds a probe that gives the largest decrease in conflict cost until no further decreases are possible. To improve the runtime, the greedy choices are made in phases, in a batched manner: in each phase, the gains for all probes are computed and then a maximal set of nonadjacent probes is selected for re-embedding by traversing the probes in nonincreasing order of gain.
2) *Chessboard [33]:* In this method, the 2-D placement grid is divided into "black" and "white" locations as in the chessboard (or checkerboard) grid of Akers [3]. The sites within each set represent a maximum independent set of locations. The Chessboard algorithm alternates between optimal re-embedding of probes placed in "black" (respectively "white") sites with respect to their neighbors (all of which are at opposite-color locations).

III. Partition Based Probe Placement

In this section, we propose a probe placement heuristic inspired from min-cut partitioning-based placement algorithms for VLSI circuits. Recursive partitioning has been the basis of numerous successful VLSI placement algorithms [5], [12], [51] since it produces placements with acceptable wire length within practical runtimes. The main goal of partitioning in VLSI is to divide a set of cells into two or four sets with minimum edge or hyperedge cut between these sets. The min-cut goal is typically achieved through the use of the Fiduccia–Mattheyses procedure [21], often in a multilevel framework [12]. Unfortunately, direct transfer of the recursive min-cut placement paradigm from VLSI to VLSIPS is blocked by the fact that the possible interactions between probes must be modeled by a complete graph and, furthermore, the border cost between two neighboring placed partitions can only be determined after the detailed placement step that finalizes probe placements at the border between the two partitions. In this section, we describe a new centroid-based quadrisection method that applies the recursive partitioning paradigm to DNA probe placement.

Assume that at a certain depth of the recursive partitioning procedure a probe set R is to be quadrisectioned into four partitions R_1, R_2, R_3, and R_4. We would like to iteratively assign each probe $p \in R$ to some partition R_i such that a minimum number of conflicts will result.[5] To approximately achieve this goal within practical runtimes, we propose to base the assignment on the number of conflicts between p and some representative, or centroid, probe $C_i \in R_i$. In our approach, for every partition R we select four centroids, one for each of the four new (sub)partitions. To achieve balanced partitions, we heuristically find four probes in R that have a maximum total distance among themselves, then use these as the centroids. This procedure, described in Fig. 3, is reminiscent of the k center approach to clustering studied by Alpert and Kahng [4] and of methods used in large-scale document classification [16].

After a given maximum partitioning depth L is reached, a final detailed placement step is needed to place each partition's probes within the partition's corresponding region on the chip. For this step, we use the Row-Epitaxial algorithm in [34] and [37], which for completeness of exposition is replicated in Fig. 4.

The complete partitioning-based placement algorithm for DNA arrays is given in Fig. 5. At a high level, our method resembles global-detailed approaches in the VLSI CAD literature [28], [45]. The algorithm recursively quadrisects every partition at a given level, assigning the probes so as to minimize distance to the centroids of subpartitions.[6] In the innermost of the three nested for loops of Fig. 5, we apply a multistart heuristic, trying r different random probes as seed C_0 and using the result that minimizes the total distance to the centroids. Once the maximum level L of recursive partitioning is attained, detailed placement is executed via the Row-Epitaxial algorithm. Additional details and commentary are as follows.

- Within the innermost of the three nested for loops, our implementation actually performs, and benefits from, a dynamic update of the partition centroid whenever a probe is added into a given partition. Intuitively, this can lead to "elongated" rather than spheric clusters, but can also correct for unfortunate choices of the initial four centroids.[7]
- The straightforward implementation of Reptx()-based detailed placement within a given partition will treat the

[5]Observe that VLSI partitioning seeks to maximize the number of nets contained within partitions (equivalently, minimize cut nets) as it assigns cells to partitions. In contrast, DNA partitioning seeks to minimize the expected number of conflicts within partitions as it assigns cells to partitions, since this leads to overall conflict reduction.

[6]The variables i and j index the row and column of a given partition within the current level's array of partitions.

[7]Details of the dynamic centroid update, reflecting an efficient implementation, are as follows. The "pseudo-nucleotide" at each position t (e.g., $t = 1, \ldots, 25$ for probes of length 25) of the centroid C_i can be represented as $C_i[t] = \bigcup_s (N_{s,t}/N_i)s$, where N_i is the current number of probes in the partition R_i, and $N_{s,t}$ is the number of probes in the partition having the nucleotide $s \in \{A, T, C, G\}$ in tth position. The Hamming distance between a probe p and C_i is $d(p, C_i) = (1/N_i) \sum_t \sum_{s \neq p[t]} N_{s,t}$.

Input: Partition (set of probes) R

Output: Probes C_0, C_1, C_2, C_3 to be used as centroids for the 4 subpartitions

---

Randomly select probe C_0 in R

Choose $C_1 \in R$ maximizing $d(C_1, C_0)$

Choose $C_2 \in R$ maximizing $d(C_2, C_0) + d(C_2, C_1)$

Choose $C_3 \in R$ maximizing $d(C_3, C_0) + d(C_3, C_1) + d(C_3, C_2)$

Return (C_0, C_1, C_2, C_3)

Fig. 3. SelectCentroid() procedure for selecting the centroid probes of subpartitions.

Input: Partition R and the neighboring partition R_n; rectangular region consisting of columns c_{left} to c_{right} and rows r_{top} to r_{bottom}

Output: Probes in R are placed in row-epitaxial fashion

Let $Q = R \cup R_n$

For $i = r_{top}$ to r_{bottom}

  For $j = c_{left}$ to c_{right}

    Find probe $q \in Q$ such that $d(q, p_{i-1,j}) + d(q, p_{i,j-1})$ is minimum

    Let $p_{i,j} = q$

    $Q = Q \setminus q$

Fig. 4. Reptx() procedure for placing a partition's probe set within the rectangular array of sites corresponding to the partition. As explained in the accompanying text, our implementation maintains the size of Q constant at $|Q| = 20\,000$ through a borrowing heuristic.

Input: Chip size $N \times N$; set R of DNA probes

Output: Probe placement which heuristically minimizes total conflicts

---

Let $l = 0$ and let $L =$ maximum recursion depth

Let $R^l_{1,1} = R$

For $l = 0$ to $L - 1$

  For $i = 1$ to 2^l

    For $j = 1$ to 2^l

      $(C_0, C_1, C_2, C_3) \leftarrow$ *SelectCentroid*$(R^l_{i,j})$

      $R^{l+1}_{2i-1,2j-1} \leftarrow \{C_0\}$; $R^{l+1}_{2i-1,2j} \leftarrow \{C_1\}$; $R^{l+1}_{2i,2j-1} \leftarrow \{C_2\}$; $R^{l+1}_{2i,2j} \leftarrow \{C_3\}$

      For each probe $p \in R^l_{i,j} \setminus \{C_0, C_1, C_2, C_3\}$

        Insert p into the yet-unfilled partition of $R^l_{i,j}$ whose centroid has minimum distance to p

For $i = 1$ to 2^L

  For $j = 1$ to 2^L

    Reptx$(R^L_{i,j}, R^L_{i,j+1})$

Fig. 5. Partitioning-based DNA probe placement heuristic.

last locations within a region "unfairly," e.g., only one candidate probe will remain available for placing in a region's last location. To ensure a uniform number of candidate probes for every position, our implementation permits "borrowing" probes from the next region in the Reptx() procedure. For every position of a region other than the last, we select the best probe from among at most m probes, where m is a predetermined constant, in the current region and the next. (Except as noted, we set m to 20 000 for all of our experiments.) Our Reptx()

TABLE I
TOTAL BORDER COST, NORMALIZED BORDER COST, GAP FROM THE SYNCHRONOUS PLACEMENT LOWER-BOUND IN [33], AND CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE TSP HEURISTIC OF [26] (TSP + 1Thr), THE ROW-EPITAXIAL (ROW-EPITAXIAL), AND SWM HEURISTICS IN [34], AND THE SIMULATED ANNEALING ALGORITHM (SA)

Chip	Lower Bound		TSP+1Thr				Row-Epitaxial				SWM				SA			
Size	Cost	Norm.	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU
100	410019	20.7	554849	28.0	35.3	113	502314	25.4	22.5	108	605497	30.6	47.7	2	583926	29.8	42.4%	20769
200	1512014	19.0	2140903	26.9	41.6	1901	1913796	24.0	26.6	1151	2360540	29.7	56.1	8	2418372	30.4	59.9%	55658
300	3233861	18.0	4667882	26.0	44.3	12028	4184018	24.0	29.4	3671	5192839	28.9	60.6	19	5502544	30.7	70.2%	103668
500	8459958	17.0	12702474	25.5	50.1	109648	11182346	22.4	32.2	10630	13748334	27.6	62.5	50	15427304	30.9	82.5%	212390

TABLE II
TOTAL/NORMALIZED BORDER COST AND CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE RECURSIVE PARTITIONING ALGORITHM WITH RECURSION DEPTH $L = 2$ AND NUMBER OF RESTARTS r VARYING FROM 1 TO 1000

Chip	RPART $r = 1$ $L = 2$			RPART $r = 10$ $L = 2$			RPART $r = 100$ $L = 2$			RPART $r = 1000$ $L = 2$		
Size	Cost	Norm.	CPU	Cost	Norm.	CPU	Cost	Norm.	CPU	Cost	Norm.	CPU
100	492054	24.9	22	491840	24.8	24	491496	24.8	47	490809	24.8	238
200	1865972	23.4	276	1865337	23.4	283	1864982	23.4	394	1864017	23.4	1064
300	4075214	22.7	1501	4074962	22.7	1527	4073135	22.7	1769	4071344	22.7	4231
500	11063382	22.2	9531	11052738	22.1	9678	11042812	22.1	13158	11039731	22.1	17014

implementation is also "border aware," that is, it takes into account Hamming distances to the placed probes in adjacent regions.

A. Empirical Evaluation of Partitioning-Based Probe Placement

In this section, we compare our partitioning-based probe placement heuristic with the TSP 1-threading heuristic in [26] (TSP + 1Thr), the Row-Epitaxial and SWM heuristics in [34], and a simulated annealing algorithm (SA).[8] We used an upper bound of 20 000 on the number of candidate probes in Row-Epitaxial and 6×6 windows with overlap 3 for SWM. The SA algorithm starts by sorting the probes and threading them onto the chip. It then slides a square window over the chip in the same way as the SWM algorithm. For every window position, SA picks two random probes in the window and swaps them with probability 1 if the swap improves the total border cost. If the swap increases border cost by δ, the swap is performed only with probability $e^{-\delta/T}$, where T is the current temperature. After experimenting with various SA parameters, we chose to run SA with 6×6 windows with overlap of 3, with 6^3 iterations performed for every window position.

Table I gives the results produced by TSP + 1Thr, Row-Epitaxial, SWM, and SA heuristics on random instances with chip sizes between 100 and 500 and probe length equal to 25. Among the four heuristics, Row-Epitaxial is the algorithm with the highest solution quality (i.e., lowest border cost) while SWM is the fastest, offering competitive solution quality with much less runtime. SA takes the largest amount of time and also gives the worse solution quality. Although it may be possible to improve SA convergence speed by extensive fine tuning of its various parameters, we expect that SA results will always remain dominated by those of the other heuristics. Additional insight into the relative quality of various heuristics can be gained by considering 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. Interestingly, for all algorithms except SA, this number decreases with increasing chip size. This can be attributed to the greater freedom of choice available when placing a higher number of probes, which all algorithms except SA seem able to exploit.

Tables II and III give results obtained by our new recursive partitioning method (RPART) with recursion depth $L = 2$ and number of restarts r varying between 1 and 1000. The results in Table II show that increasing the number of restarts gives a small improvement in border cost at the expense of increased runtime. Table III presents results obtained by RPART when run with $r = 10$ for recursion depth L varying between 1 and 3. Comparing to the results produced by Row-Epitaxial, the best heuristic from Table I, we find that recursive partitioning-based placement achieves on the average similar or better results with improved runtime.

We next discuss in more detail the runtime of RPART, which is somehow unusual for a recursive partitioning algorithm in that it may get smaller with an increase in recursion depth. Let the number of probes in a chip be n. The two main contributors to RPART runtime are the recursive partitioning phase, whereby the probes are divided into smaller and smaller partition regions, and the detailed placement step, which is achieved by running Row-Epitaxial within each partition region (with borrowing from next region when needed). Since

[8]All experiments reported in this paper were performed on test cases obtained by generating each probe candidate uniformly at random. The probe length was set to 25, which is the typical value for commercial arrays [1]. Unless otherwise noted, we used the canonical periodic deposition sequence $(ACTG)^{25}$. All reported runtimes are for a 2.4-GHz Intel Xeon server with 2 GB of RAM running under Linux.

TABLE III
TOTAL BORDER COST, NORMALIZED BORDER COST, GAP FROM THE SYNCHRONOUS PLACEMENT LOWER-BOUND IN [33], AND CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE RECURSIVE PARTITIONING ALGORITHM WITH RECURSION DEPTH VARYING BETWEEN ONE AND THREE

Chip	Lower Bound		RPART $r = 10$ $L = 1$				RPART $r = 10$ $L = 2$				RPART $r = 10$ $L = 3$			
Size	Cost	Norm.	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU
100	410019	20.7	475990	24.0	16.1	69	491840	24.8	20.0	24	504579	25.5	23.1	10
200	1512014	19.0	1813105	22.8	19.9	992	1865337	23.4	23.4	283	1922951	24.2	27.2	81
300	3233861	18.0	4135728	23.8	27.9	3529	4074962	22.7	26.0	1527	4175146	24.0	29.1	240
500	8459958	17.0	11283631	22.6	33.4	10591	11052738	22.1	30.6	9678	11134960	22.3	31.6	3321

TABLE IV
TOTAL BORDER COST, NORMALIZED BORDER COST, GAP FROM THE ASYNCHRONOUS POSTPLACEMENT LOWER BOUND IN [33], AND CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE BATCHED GREEDY, CHESSBOARD, AND SEQUENTIAL IN-PLACE RE-EMBEDDING ALGORITHMS

Chip	Lower Bound		Batched Greedy				Chessboard				Sequential			
Size	Cost	Norm.	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU
100	364953	18.4	458746	23.2	25.7	40	439768	22.2	20.5	54	437536	22.1	19.9	64
200	1425784	17.9	1800765	22.6	26.3	154	1723773	21.7	20.9	221	1715275	21.5	20.3	266
300	3130158	17.4	3965910	22.1	26.7	357	3803142	21.2	21.5	522	3773730	21.0	20.6	577
500	8590793	17.2	10918898	21.9	27.1	943	10429223	20.9	21.4	1423	10382620	20.8	20.9	1535

executing the procedure SelectCentroid() and distributing all the probes in a partition region to its four subregions take the time proportional to the number of probes in a region, the total runtime for each recursion depth is $O(n)$ and the overall runtime for the recursive partitioning phase is $O(Ln)$. Clearly, this component of the runtime increases linearly with recursion depth. On the other hand, in our implementation of RPART, the Row-Epitaxial algorithm used in detailed placement considers at most $\min\{m, 2n/4^L\}$ candidate probes for placement at any given position ($m = 20\,000$ is a predetermined upper bound that we impose based on the empirical results in [34] and $2n/4^L$ is a bound that follows from the fact that we never consider candidates from more than two consecutive lowest-level partition regions). Thus, the total time needed by the detailed placement step is $O(n \min\{m, 2n/4^L\})$, which will decrease with increasing L once L exceeds $\log_4(2n/m)$. This explains why the overall RPART runtime in Table III decreases with increasing L, and also explains why solution quality may slightly degrade with increasing L due to the reduced number of probe candidates considered by Row-Epitaxial for each chip location.

B. Comparison of Complete Probe Placement and Embedding Flows

In addition to the partitioning-based placement algorithm, we propose a new algorithm that performs optimal re-embedding of probes in a sequential row-by-row fashion. We believe that a main shortcoming of Batched Greedy and Chessboard (described in Section II-D) is that these methods always re-embed an independent set of sites on the DNA chip. Dropping this requirement permits faster propagation of the effects of any re-embedding decision.

Table IV compares the new probe embedding algorithm with Batched Greedy and Chessboard on random instances with chip sizes between 100 and 500 and probe length 25 for which the 2-D placements were obtained using TSP + 1-threading. All algorithms are stopped when the improvement cost achieved in one iteration over the whole chip drops below 0.1% of the total cost. The results show that re-embedding of the probes in a sequential row-by-row order leads to reduced border cost with similar runtime compared to previous methods.

In another series of experiments, we ran complete placement and embedding flows obtained by combining each of the five 2-D placement algorithms evaluated in Section III-A with the sequential in-place re-embedding algorithm. Results are given in Tables V and VI. Again, SA and TSP + 1Thr are dominated by both REPTX and SWM in both conflict cost and running time. REPTX produces less conflicts than SWM, but SWM is considerably faster. Recursive partitioning consistently outperforms the best previous flow (Row-Epitaxial + sequential re-embedding)—by an average of 4.0%—with similar or lower runtime.

IV. FLOW ENHANCEMENTS

The current DNA array design flow can be significantly improved by introducing flow-aware problem formulations, adding feedback loops between optimization steps, and/or integrating multiple optimizations. These enhancements, which are represented schematically in Fig. 6 by the dashed arcs, are similar to flow enhancements that have proved very effective in the VLSI design context [19], [48].

In this paper, we concentrate on two such enhancements, both aiming at further reductions in total border length. The first enhancement is a tighter integration between probe placement

TABLE V
TOTAL BORDER COST, NORMALIZED BORDER COST, GAP FROM THE ASYNCHRONOUS PREPLACEMENT LOWER BOUND IN [33], AND CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE TSP HEURISTIC OF [26] (TSP + 1Thr), THE ROW-EPITAXIAL (ROW-EPITAXIAL), AND SWM HEURISTICS OF [34], AND THE SIMULATED ANNEALING ALGORITHM (SA)

Chip	Lower Bound		TSP+1Thr				Row-Epitaxial				SWM				SA			
Size	Cost	Norm.	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU
100	220497	11.1	439829	22.2	99.5	113	415227	21.0	88.3	161	440648	22.3	99.8	93	457752	23.1	107.6	11713
200	798708	10.0	1723352	21.7	115.8	1901	1608382	20.2	101.4	1368	1721633	21.6	115.6	380	1844344	23.2	130.9	42679
300	—	—	3801765	21.2	—	12028	3529745	20.3	—	3861	3801479	21.2	—	861	4155240	23.2	—	101253
500	—	—	10426237	20.9	—	109648	9463941	19.0	—	12044	10161979	20.4	—	2239	11574482	23.2	—	222376

TABLE VI
TOTAL BORDER COST, NORMALIZED BORDER COST, GAP FROM THE ASYNCHRONOUS PREPLACEMENT LOWER-BOUND IN [33], AND CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE RECURSIVE PARTITIONING ALGORITHM FOLLOWED BY SEQUENTIAL IN-PLACE RE-EMBEDDING

Chip	Lower Bound		RPART $r = 10$ $L = 1$				RPART $r = 10$ $L = 2$				RPART $r = 10$ $L = 3$			
Size	Cost	Norm.	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU	Cost	Norm.	Gap(%)	CPU
100	220497	11.1	393218	19.9	78.3	123	399312	20.2	81.1	44	410608	20.7	86.2	10
200	798708	10.0	1524803	19.2	90.9	1204	1545825	19.4	93.5	365	1573096	19.8	97.0	101
300	—	—	3493552	20.1	—	3742	3413316	19.6	—	1951	3434964	19.7	—	527
500	—	—	9546351	19.1	—	11236	9355231	18.8	—	10417	9307510	18.7	—	3689

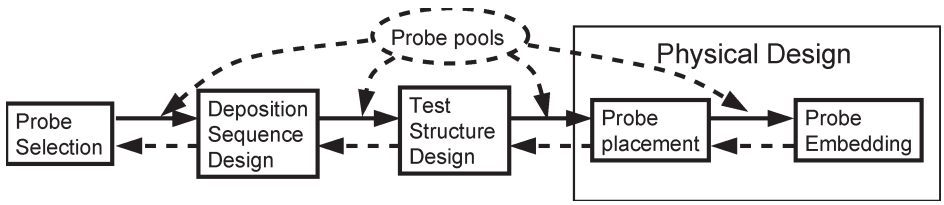


Fig. 6. Typical DNA array design flow with solid arcs and proposed enhancements represented by dashed arcs.

and embedding; this enhancement is discussed in Section IV-B. The second enhancement is the integration between physical design and probe selection, which is achieved by passing the entire pools of candidates available for each probe to the physical design step. As shown in Section IV-C, this enhancement enables significant improvements (up to 15%) in border length compared to the best previous flows [34], [37].

Other feedback loops and integrated optimizations are possible but are not explored in this paper. Faster and more targeted probe selection may be achievable by adding a feedback loop to provide updated selection rules and parameters to the probe selection step. Integrating deposition sequence design with probe selection may lead to further reductions in the number of masks by exploiting the freedom available in choosing the candidates for each probe.

A. Problem Formulation for Integrated Probe Selection and Physical Design

To integrate probe selection and physical design, we pass the entire pools of candidates for each probe to the physical design step (Fig. 6). As discussed in Section II-A, all probe candidates are selected so that they have similar hybridization properties (e.g., melting temperatures) and can thus be used interchangeably. The availability of multiple probe candidates gives additional freedom during placement and embedding, and may potentially reduce final border cost. DNA array physical design with probe pools is captured by the following problem formulation.[9]

Integrated DNA Array Design Problem

Given:

- pools of candidates $P_i = \{p_{ij} | j = 1, \ldots, l_i\}$ for each probe $i = 1, \ldots, N^2$, where $N \times N$ is the size of array;
- number of masks K.

Find:

1) probes $p_{ij} \in P_i$ for every $i = 1, \ldots, N^2$;
2) a deposition sequence $S = s_1, \ldots, s_K$ that is a supersequence of all selected probes p_{ij};
3) a placement of the selected probes p_{ij} into an $N \times N$ array;
4) an embedding of the selected probes p_{ij} into the deposition sequence S.

Such that:

- total number of conflicts between adjacent embedded probes is minimized.

[9]This formulation also integrates deposition sequence design. For simplicity, we leave out design of control and test sequences.

Although the flow in Fig. 6 suggests a particular order for making the choices 1–4, the integrated formulation above allows interleaving these decisions. The following two algorithms capture key optimizations in the integrated formulations and are used as core building blocks in the solution methods evaluated in Sections IV-B and IV-C. They are "probe pool" versions of the Row-Epitaxial and re-embedding algorithms proposed in [34] and [37], and degenerate to the latter ones in the case when each probe pool contains a single candidate.

- The Pool Row-Epitaxial algorithm (Pool-REPTX) is the extension to probe pools of the REPTX probe placement algorithm in [34] and [37]. Pool-REPTX performs choices 1 and 3 for given choices 2 and 4, i.e., it simultaneously chooses already embedded candidates from the respective pools and places them in the $N \times N$ array. The input of Pool-REPTX consists of probe candidates p_{ij} embedded in the deposition sequence S. Each such embedding is written as a sequence of length $K = |S|$ over the alphabet $\{A, C, T, G, \text{Blank}\}$, where A, C, T, and G denote embedded nucleotides and Blanks denote positions of S left unused by the embedded candidate probe. Pool-REPTX consists of the following steps: 1) lexicographic sorting of the pools (based on the first candidate, when more than one candidate is available in the pool); 2) threading the sorted pools in row-by-row order into the $N \times N$ array; 3) finding, in row-by-row order, the best probe candidate—i.e., the candidate having the minimum number of conflicts with already placed neighbors—among the not yet placed pools within a prescribed lookahead region.
- The sequential in-place pool re-embedding algorithm is the extension to probe pools of the sequential probe re-embedding algorithm given in Section III-B. It complements Pool-REPTX by iteratively modifying candidate selections within each pool and their embedding (choices 2 and 4) as follows. In row-by-row order, for each position in the $N \times N$ array, and for each candidate p_{ij} from the pool of the respective probe, an embedding having a minimum number of conflicts with the existing embeddings of the neighbors is computed, and then the best embedded candidate replaces the current one.

B. Improved Integration of Probe Placement and Embedding

As noted in [33], allowing arbitrary, or asynchronous, embeddings leads to further reductions in border length compared to synchronous embedding [e.g., contrast Fig. 2(b) and (c)]. An interesting question is finding the best order in which the placement and embedding degrees of freedom should be exploited. Previous methods [33], [34], [37] can be divided into two classes: 1) methods that perform placement and embedding decisions simultaneously, and 2) methods that exploit the two degrees of freedom one at a time. Currently, best methods in the second class (e.g., synchronous Row-Epitaxial followed by chessboard/sequential in-place probe re-embedding [34], [37]) outperform the methods in the first class (e.g., the asynchronous epitaxial algorithm in [33]) in terms of both runtime and solution quality.

All known methods in the second class perform synchronous probe placement followed by iterated in-place re-embedding of the probes (with locked probe locations). More specifically, these methods perform the following three steps:

1) synchronous embedding of the probes;
2) probe placement with costs given by the Hamming distance between synchronous probe embeddings;
3) iterated sequential probe re-embedding.

We note that significant reductions in border cost are possible by performing the placement based on asynchronous, rather than synchronous, embeddings of the probes, and therefore modify the above scheme as follows:

1) asynchronous embedding of the probes;
2) placement with costs given by the Hamming distance between fixed asynchronous probe embeddings;
3) iterated sequential probe re-embedding.

Since solution spaces for placement and embedding are still searched independently of one another and the computation of an initial asynchronous embedding does not add significant overhead, the proposed change is unlikely to adversely affect the runtime. However, because placement optimization is now applied to embeddings more similar to those sought in the final optimization stage, there is significant potential for improvement.

In the current embodiment of the modified scheme, we implement the first step by using for each probe the "as soon as possible," or *ASAP*, embedding [see Fig. 2(c)]. Under ASAP embedding, the nucleotides in a probe are embedded sequentially by always using the earliest available synthesis step. The intuition behind using ASAP embeddings is that, since ASAP embeddings are more densely packed, the likelihood that two neighboring probes will both use a synthesis step increases compared to synchronous embeddings. This translates directly into reductions in the number of border conflicts.

Indeed, consider two random probes p and p' picked from uniform distribution. When we perform synchronous embedding, the length of the deposition sequence is $4 \times 25 = 100$. The probability that any one of the 100 synthesis steps is used by one of the random probes and not the other is $2 \times (1/4) \times (3/4)$, and therefore the expected number of conflicts is $100 \times 2 \times (1/4) \times (3/4) = 37.5$. Assume now that the two probes are embedded using the ASAP algorithm. Notice that for every $0 \leq i \leq 3$ the ASAP algorithm will leave a gap of length i with probability 1/4 between any two consecutive letters of a random probe. This results in an average gap length of 1.5 and an expected number of synthesis steps of $25 + 24 * 1.5 = 61$. Assuming that p and p' are both embedded within 61 steps, the number of conflicts between their ASAP embeddings is then approximately $61 \times 2 \times (25/61) \times ((61 - 25)/61) \approx 29.5$. Although in practice many probes require more than 61 synthesis steps when embedded using the ASAP algorithm, they still require much less than 100 steps and result in significantly fewer conflicts compared to synchronous embedding.

To empirically evaluate the advantages of ASAP embedding, we compared test cases ranging in size from 100×100 to 500×500 the "champion" method in [33], [34], and [37], which uses synchronous initial embeddings for the

TABLE VII
TOTAL BORDER COST (AVERAGES OVER TEN RANDOM INSTANCES) FOR SYNCHRONOUS AND ASAP INITIAL PROBE EMBEDDING FOLLOWED BY ROW-EPITAXIAL AND ITERATED SEQUENTIAL IN-PLACE PROBE RE-EMBEDDING

Chip	Synchronous Initial Embedding			ASAP Initial Embedding			Percent
	Sync Embed	REPTX	Re-Embed	ASAP Embed	REPTX	Re-Embed	Improv.
100	619153	502314	415227	514053	393765	389637	5.2
200	2382044	1918785	1603745	1980913	1496937	1484252	6.7
300	5822857	4193439	3514087	4357395	3273357	3245906	6.9
500	18786229	11203933	9417723	11724292	8760836	8687596	7.0

TABLE VIII
CPU SECONDS (AVERAGES OVER TEN RANDOM INSTANCES) FOR SYNCHRONOUS AND ASAP INITIAL PROBE EMBEDDING FOLLOWED BY ROW-EPITAXIAL AND ITERATED SEQUENTIAL IN-PLACE PROBE RE-EMBEDDING

Chip	Synchronous Initial Embedding			ASAP Initial Embedding		
	Sync+REPTX	Re-Embed	Total	ASAP+REPTX	Re-Embed	Total
100	166	81	247	188	29	217
200	1227	340	1567	1302	114	1416
300	3187	748	3935	2736	235	2971
500	8495	2034	10529	6391	451	6842

probes, with the corresponding method based on ASAP initial probe embeddings. For both methods, the second and third steps are implemented using REPTX and sequential in-place probe re-embedding algorithms in [34] and [37] (see, also, Section IV-A).

Tables VII and VIII give the border length and CPU time (in seconds) for the two methods. Each number in these tables represents the average of over ten test cases of the given size. Surprisingly, the simple switch from synchronous to ASAP initial embedding results in 5%–7% reduction in total border length. Furthermore, the runtimes for the two methods are comparable. In fact, sequential re-embedding becomes faster in the ASAP-based method compared to the synchronous-based one since fewer iterations are needed to converge to a locally optimal solution (the number of iterations drops from nine to three on the average).

C. Integrated Probe Selection and Physical Design

We explored two methods for exploiting the availability of multiple probe candidates during placement and embedding. The first method uses the Row-Epitaxial and sequential in-place probe re-embedding algorithms described in Section IV-A. This method is an instance of integration between multiple flow steps since probe selection decisions are made during probe placement and can be further changed during probe re-embedding. The detailed steps are as follows:

- perform ASAP embedding of all probe candidates;
- run the Pool-REPTX or a pool version of the recursive-partitioning placement algorithm in Section III using border costs given by the Hamming distance between the ASAP embeddings;
- run the sequential in-place pool re-embedding algorithm.

The second method preserves the separation between candidate selection and placement + embedding. However, we modify probe selection to make it flow aware, i.e., to make its results more suitable for the subsequent placement and embedding optimizations. Building on the observation that shorter probe embeddings lead to improved border length, we choose from the available candidates the one that embeds in the least number of steps of the standard periodic deposition sequence using ASAP:

- perform ASAP embedding of all probe candidates;
- select from each pool of candidates the one that embeds the least number of steps using ASAP;
- run the REPTX or recursive-partitioning placement algorithm using only the selected candidates and border costs given by the Hamming distance between ASAP embeddings;
- run the iterated sequential in-place probe re-embedding algorithm, again using only selected candidates.

Table IX gives the border length and the runtime (in CPU seconds) for the two methods of combining probe placement and embedding with probe selection (each number represents the average of over ten test cases of the given size). We report results for both the Pool-REPTX placement algorithm and the pool version of the recursive partitioning using $L = 3$. We varied the number of candidates available for each probe between 1 and 16; probe candidates were generated uniformly at random.

As expected, for each method and chip size, the improvement in solution quality grows monotonically with the number of available candidates. The improvement is significant (up to 15% when running the first method on a 100×100 chip with 16 candidates per probe) but varies nonuniformly with the method and chip size. For small chips, the first method gives better

TABLE IX
TOTAL BORDER COST AND RUNTIME (AVERAGES OVER TEN RANDOM INSTANCES) FOR THE TWO METHODS OF COMBINING PROBE PLACEMENT AND EMBEDDING WITH PROBE SELECTION. IMPROVEMENT (IN PERCENT) IS RELATIVE TO THE SINGLE-CANDIDATE VERSION OF THE SAME CODE. WE REPORT RESULTS FOR BOTH THE REPTX ALGORITHM AND THE RECURSIVE-PARTITIONING ALGORITHM WITH $L = 3$

Chip	Pool	Multi-Candidate						ASAP-Based Selection					
		Row-Epitaxial			Partitioning			Row-Epitaxial			Partitioning		
Size	Size	Border	CPU	%	Border	CPU	%	Border	CPU	%	Border	CPU	%
100	1	389637	217	–	376348	115	–	389637	217	–	376348	115	–
	2	372951	1040	4.3	372957	676	0.9	377026	212	3.2	362882	114	3.6
	4	357562	1796	8.2	357553	1274	5.0	363944	193	6.6	350079	127	7.0
	8	343604	3645	11.8	343590	2605	8.7	351540	191	9.8	341020	109	9.4
	16	330600	7315	15.2	330551	5003	12.2	339636	185	12.8	332634	121	11.6
200	1	1484252	1416	–	1446489	1012	–	1484252	1416	–	1446498	1012	
	2	1438182	6278	3.1	1438345	7281	0.6	1435712	1176	3.3	1410533	946	2.5
	4	1386527	12750	6.6	1386424	13231	4.1	1385556	1189	6.6	1361653	932	5.9
	8	1334273	27382	10.1	1334519	26413	7.7	1336851	1121	9.9	1313294	957	9.2
	16	1284550	44460	13.5	1284462	52400	11.2	1289566	1117	13.1	1276855	971	11.7
300	1	3245906	2971	–	3220850	2975	–	3245906	2971	–	3220850	2975	–
	2	3185015	14956	1.9	3184426	13161	1.1	3141088	2724	3.2	3046086	2134	5.4
	4	3093633	26514	4.7	3093944	24671	3.9	3018490	2771	7.0	2936733	2118	8.8
	8	2985393	51226	8.0	2986286	45607	7.3	2921195	2603	10.0	2832535	2079	12.0
	16	2878886	98189	11.3	2878244	85311	10.6	2835695	2760	12.6	2706537	2247	15.9
500	1	8687596	6842	–	8645162	5608	–	8687596	6842	–	8645162	5608	–
	2	8611468	51847	0.9	8611142	41409	0.4	8407839	6090	3.2	8273184	5468	4.3
	4	8477014	86395	2.4	8479150	94566	1.9	8105358	6709	6.7	7955391	5591	8.0
	8	8248838	161651	5.1	8249176	213264	4.6	7807763	6085	10.1	7637927	5782	11.7
	16	–	–	–	–	–	–	7518331	5986	13.5	7445283	5601	13.9

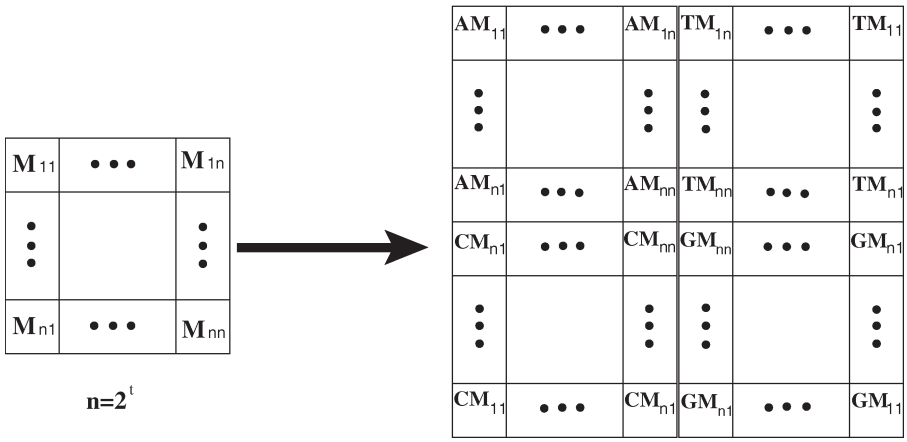


Fig. 7. Two-dimensional Gray code placement.

solution quality than the second. For chips of size 200 × 200, the two methods give comparable solution quality, while for chips with size 300 × 300 or larger the second method is better (by over 5% for 500 × 500 chips with eight probe candidates). The second method is faster than the first for all chip sizes. The speedup factor varies between $5\times$ and $40\times$ when the number of candidates varies between 2 and 16. Interestingly, the runtime of the second method is slightly improving with the number of candidates, the reason being that the number of iterations of sequential re-embedding decreases when the length of the ASAP embedding of the selected candidates decreases.

V. QUANTIFIED SUBOPTIMALITY OF PLACEMENT AND EMBEDDING ALGORITHMS

As noted in the Introduction, next-generation DNA probe arrays will contain up to 100 million probes and therefore present instance complexities for placement that will far outstrip those of VLSI designs. Thus, it is of interest to study not only runtime scaling but also scaling of suboptimality for available heuristics. To this end, we apply the experimental framework for quantifying suboptimality of placement heuristics that was originated by Boese and by Hagen *et al.* [27], and recently extended by Chang *et al.* [13] and Cong *et al.* [15]. In this

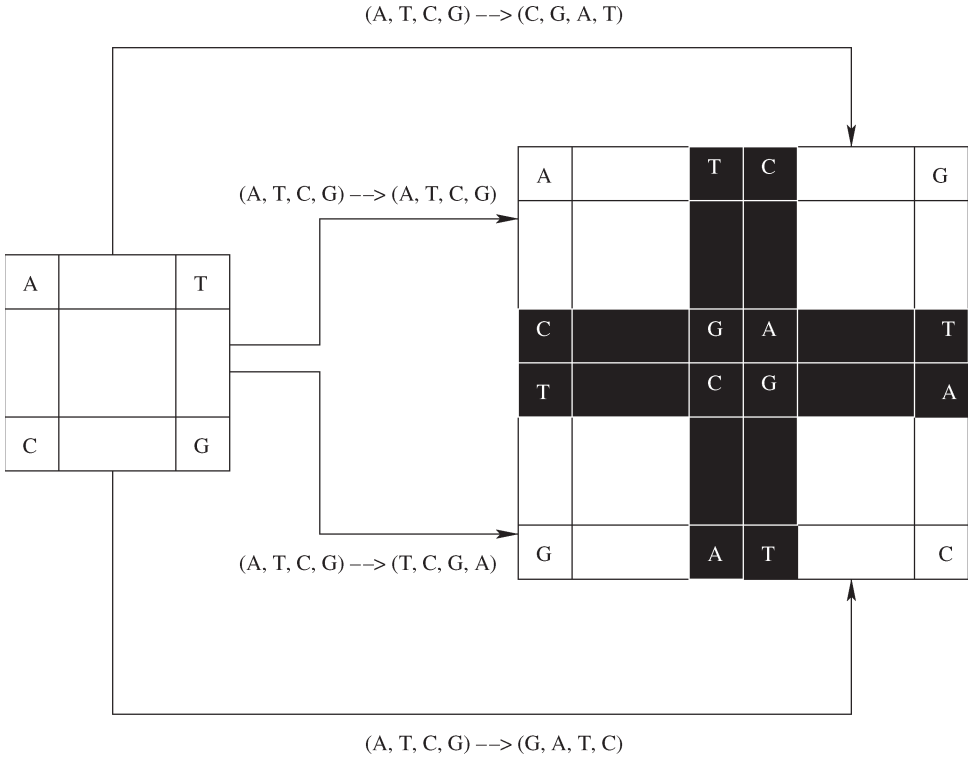


Fig. 8. Scaling construction used in the suboptimality experiment.

framework, there are two basic types of instance scaling that we can apply.

- Instances with known optimum solution. For hypergraph placement, instances with known minimum wire length solutions may be constructed by "overlaying" signal nets within an already placed cell layout such that each signal net has probably minimum length. This technique, proposed by Boese and further explored by Chang *et al.* [13], induces a netlist topology with prescribed degree sequence over the (placed) cells; this corresponds to a "placement example with known optimal wire length" (PEKO). In our DNA probe placement context, there is no need to generate a netlist hypergraph. Rather, we realize the concept of minimum (border) cost edges (adjacencies) by constructing a set of probes, and their placement, using 2-D Gray codes [20]. Our construction generates 4^k probes that are placeable such that every probe has a border cost of 2 to each of its neighboring probes. This construction is illustrated in Fig. 7.
- Instances with known suboptimal solutions. Because constructed instances with known optimum solutions may not be representative of "real" instances, we also apply a technique [27] that allows real instances to be scaled such that they offer insights into the scaling of heuristic suboptimality. The technique is applied as follows. Beginning with a problem instance I, we construct three isomorphic versions of I by three distinct mappings of the nucleotide set $\{A, C, G, T\}$ onto itself. Each mapping yields a new probe set that can be placed with optimum border cost exactly equal to the optimum border cost of I. Our scaled instance I' consists of the union of the original probe set and its three isomorphic copies. Observe that one placement solution for I' is to optimally place I and its isomorphic copies as individual chips, and then to adjoin these placements as the four quadrants of a larger chip. Thus, an upper bound on the optimum border cost for I' is four times the optimum border cost for I plus the border cost between the copies of I (see Fig. 8). If a heuristic H places I' with cost $c_H(I') \geq 4c_H(I)$, then we may infer that the heuristic's suboptimality is growing by at least a factor $(c_H(I')/4c_H(I))$. On the other hand, if $c_H(I') < 4c_H(I)$, then the heuristic's solution quality would be said to scale well on this class of instances.

Table X shows results from executing the various placement heuristics on PEKO-style test cases, with instance sizes ranging from 16×16 to 512×512 (recall that our Gray code construction yields instances with 4^k probes). We see from these results that SWM is closest to the optimum, with a suboptimality gap of 4%–30%. Overall, DNA array placement algorithms appear to be performing better than their VLSI counterparts [13] when it comes to results on special case instances with known optimal cost. Of course, results from placement algorithms (whether for VLSI or DNA chips) on special benchmark instances should not be generalized to arbitrary benchmarks. In particular, our results show that algorithms that perform best for arbitrary benchmarks are not necessarily the best performers for specially constructed benchmarks.

Table XI shows results from executing the various placement heuristics on scaled versions of random DNA probe sets, with the original instances ranging in size from 100×100 to 500×500, and the scaled instances thus ranging in size from 200×200 to 1000×1000. This table shows that, in general,

TABLE X
COMPARING THE PLACEMENT ALGORITHM PERFORMANCE FOR CASES WITH KNOWN OPTIMAL CONFLICTS. SW MATCHING IS USING A WINDOW SIZE OF 20 × 20 AND A STEP OF 10. ROW-EPITAXIAL USES 10 000/CHIP SIZE LOOKAHEAD ROWS

Chip Size	Optimal Cost	TSP+Threading		Row-Epitaxial		SWM		Recursive Partitioning	
		Cost	Gap(%)	Cost	Gap(%)	Cost	Gap(%)	Cost	Gap(%)
16	960	1380	44	960	0	992	4	1190	24
32	3968	6524	65	5142	30	4970	25	5210	31
64	16128	27072	68	16128	0	19694	22	21072	31
128	65024	111420	71	92224	42	86692	33	88746	36
256	261120	457100	75	378612	45	325566	25	359060	37
512	1046528	1844244	76	1573946	50	1414154	35	1476070	41

TABLE XI
COMPARING THE SUBOPTIMALITY OF THE PLACEMENT ALGORITHMS' PERFORMANCE FOR VARIOUS BENCHMARKS. EACH ENTRY REPRESENTS BOTH THE UPPER BOUND AND THE ACTUAL PLACEMENT RESULT AFTER SCALING. SW MATCHING IS USING A WINDOW SIZE OF 20 × 20 AND A STEP OF 10. ROW-EPITAXIAL USES 10 000/CHIP SIZE LOOKAHEAD ROWS

Instance Size	Row Epitaxial			SW-Matching			Recursive Partitioning		
	U-Bound	Actual	Ratio	U-Bound	Actual	Ratio	U-Bound	Actual	Ratio
100	2024464	1479460	0.73	2203132	1999788	0.91	1919328	1425806	0.73
200	7701848	6379752	0.83	8478520	6878096	0.81	7497520	6107394	0.82
300	16817110	12790186	0.76	18645122	13957686	0.75	16699806	12567786	0.75
400	29239934	24621324	0.84	32547390	26838164	0.82	30450780	24240850	0.80
500	44888710	38140882	0.85	49804320	41847206	0.84	47332142	37811712	0.80

placement algorithms for DNA arrays offer excellent scaling suboptimality. We believe that this is primarily due to the already noted fact that algorithm quality (as reflected by normalized border costs) improves with instance size. The larger number of probes in the scaled instances gives more freedom to the placement algorithms, leading to heuristic placements that have scaling suboptimality factor well below 1.

VI. CONCLUSION

In this paper, we have studied several problems arising in the design of DNA chips, focusing on minimizing the total border length between adjacent sites during probe placement and embedding. We have shown that significant reductions in border length can be obtained by drawing algorithmic techniques developed in the field of VLSI design automation.

We conclude with some remarks on the similarities and differences between VLSI physical design and physical design for DNA arrays. First, while VLSI placement performance in general degrades as the problem size increases, it appears that this is not the case for DNA array placement. Current algorithms are able to find DNA array placements with smaller normalized border cost when the number of probes in the design grows. Second, the lower bounds for DNA probe placement and embedding appear to be tighter than those available in the VLSI placement literature. Developing even tighter lower bounds is, of course, an important open problem.

Another direction of future research is to find formulations and methods for integrated optimization of test structure design and physical design. Since test structures are typically preplaced at sites uniformly distributed across the array, integrated optimization can have a significant impact on the total border length.

REFERENCES

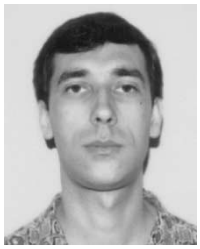
[1] Online tutorial, Affymetrix, Inc. [Online]. Available: http://www.affymetrix.com/technology/
[2] Online tutorial, Perlegen Sciences, Inc. [Online]. Available: http://www.perlegen.com/science/arrays.htm
[3] S. Akers, "On the use of the linear assignment algorithm in module placement," in *Proc. ACM/IEEE Design Automation Conf. (DAC)*, Nashville, TN, 1981, pp. 137–144.
[4] C. J. Alpert and A. B. Kahng, "Geometric embeddings for faster (and better) multi-way netlist partitioning," in *Proc. ACM/IEEE Design Automation Conf.*, Dallas, TX, 1993, pp. 743–748.
[5] ——, "Recent directions in netlist partitioning: A survey," *Integration: VLSI J.*, vol. 19, no. 1–2, pp. 1–81, Aug. 1995.
[6] N. Alon, C. J. Colbourn, A. C. H. Lingi, and M. Tompa, "Equireplicate balanced binary codes for oligo arrays," *SIAM J. Discrete Math.*, vol. 14, no. 4, pp. 481–497, 2001.
[7] A. A. Antipova, P. Tamayo, and T. R. Golub, "A strategy for oligonucleotide microarray probe reduction," *Genome Biol.*, vol. 3, no. 12, pp. research0073.1–research0073.4, Nov. 2002.
[8] M. Atlas, N. Hundewale, L. Perelygina, and A. Zelikovsky, "Consolidating software tools for DNA microarray design and manufacturing," in *Proc. Int. Conf. IEEE Engineering Medicine and Biology (EMBC)*, San Francisco, CA, 2004, pp. 172–175.
[9] K. Nandan Babu and S. Saxena, "Parallel algorithms for the longest common subsequence problem," in *Proc. 4th Int. Conf. High-Performance Computing*, Bangalore, India, Dec. 1997, pp. 120–125.
[10] J. Branke and M. Middendorf, "Searching for shortest common supersequences," in *Proc. 2nd Nordic Workshop Genetic Algorithms and Applications*, Vaasa, Finland, 1996, pp. 105–113.
[11] J. Branke, M. Middendorf, and F. Schneider, "Improved heuristics and a genetic algorithm for finding short supersequences," *OR Spektrum*, vol. 20, no. 1, pp. 39–46, Feb. 1998.

[12] A. Caldwell, A. Kahng, and I. Markov, "Can recursive bisection produce routable designs?," in *Proc. Design Automation Conf. (DAC)*, Los Angeles, CA, 2000, pp. 477–482.
[13] C. C. Chang, J. Cong, and M. Xie, "Optimality and scalability study of existing placement algorithms," in *Proc. Asia South-Pacific Design Automation Conf.*, Kitakyushu, Japan, Jan. 2003, pp. 621–627.
[14] C. J. Colbourn, A. C. H. Lingi, and M. Tompa, "Construction of optimal quality control for oligo arrays," *Bioinformatics*, vol. 18, no. 4, pp. 529–535, Apr. 2002.
[15] J. Cong, M. Romesis, and M. Xie, "Optimality, scalability and stability study of partitioning and placement algorithms," in *Proc. Int. Symp. Physical Design (ISPD)*, Monterey, CA, 2003, pp. 88–94.
[16] D. R. Cutting, D. R. Karger, J. O. Pederson, and J. W. Tukey, "Scatter/gather: A cluster-based approach to browsing large document collections," in *Proc. 15th Int. ACM/SIGIR Conf. Research and Development Information Retrieval SIGIR Forum*, Copenhagen, Denmark, 1992, pp. 318–329.
[17] V. Dancik, "Common subsequences and supersequences and their expected length," *Comb. Probab. Comput.*, vol. 7, no. 4, pp. 365–373, Dec. 1998.
[18] K. Doll, F. M. Johannes, and K. J. Antreich, "Iterative placement improvement by network flow methods," *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.*, vol. 13, no. 10, pp. 1189–1200, Oct. 1994.
[19] J. J. Engel *et al.*, "Design methodology for IBM ASIC products," *IBM J. Res. Develop.*, vol. 40, no. 4, p. 387, Jul. 1996.
[20] W. Feldman and P. A. Pevzner, "Gray code masks for sequencing by hybridization," *Genomics*, vol. 23, no. 1, pp. 233–235, Sep. 1994.
[21] C. M. Fiduccia and R. M. Mattheyses, "A linear-time heuristic for improving network partitions," in *Proc. Design Automation Conf. (DAC)*, Las Vegas, NV, 1982, pp. 175–181.
[22] S. Fodor, J. L. Read, M. C. Pirrung, L. Stryer, L. A. Tsai, and D. Solas, "Light-directed, spatially addressable parallel chemical synthesis," *Science*, vol. 251, no. 4995, pp. 767–773, Feb. 1991.
[23] D. E. Foulser, M. Li, and Q. Yang, "Theory and algorithms for plan merging," *Artif. Intell.*, vol. 57, no. 2–3, pp. 143–181, Oct. 1992.
[24] C. B. Fraser and R. W. Irving, "Approximation algorithms for the shortest common supersequence," *Nord. J. Comput.*, vol. 2, no. 3, pp. 303–325, 1995.
[25] D. H. Geschwind and J. P. Gregg, Eds. *Microarrays for the Neurosciences: An Essential Guide*. Cambridge, MA: MIT Press, 2002.
[26] S. Hannenhalli, E. Hubbell, R. Lipshutz, and P. A. Pevzner, "Combinatorial algorithms for design of DNA arrays," in *Chip Technology*, J. Hoheisel, Ed. New York: Springer-Verlag, 2002.
[27] L. W. Hagen, D. J. Huang, and A. B. Kahng, "Quantified suboptimality of VLSI layout heuristics," in *Proc. ACM/IEEE Design Automation Conf.*, San Francisco, CA, 1995, pp. 216–221.
[28] D. J. Huang and A. B. Kahng, "Partitioning-based standard-cell global placement with an exact objective," in *Proc. ACM/IEEE Int. Symp. Physical Design*, Napa, CA, Apr. 1997, pp. 18–25.
[29] E. Hubbell and P. A. Pevzner, "Fidelity probes for DNA arrays," in *Proc. 7th Int. Conf. Intelligent Systems Molecular Biology*, Heidelberg, Germany, 1999, pp. 113–117.
[30] E. Hubbell and M. Mittman, Private Communication, Jul. 2002.
[31] T. Jiang and M. Li, "On the approximation of shortest common supersequences and longest common subsequences," *SIAM J. Discrete Math.*, vol. 24, no. 5, pp. 1122–1139, Oct. 1995.
[32] L. Kaderali and A. Schliep, "Selecting signature oligonucleotides to identify organisms using DNA arrays," *Bioinformatics*, vol. 18, no. 10, pp. 1340–1349, Oct. 2002.
[33] A. B. Kahng, I. I. Măndoiu, P. A. Pevzner, S. Reda, and A. Zelikovsky, "Border length minimization in DNA array design," in *Proc. 2nd Int. Workshop Algorithms Bioinformatics (WABI)*, Rome, Italy, R. Guigó and D. Gusfield, Eds., vol. 2452, 2002, pp. 435-448.
[34] ——, "Engineering a scalable placement heuristic for DNA probe arrays," in *Proc. 7th Annu. Int. Conf. Research Computational Molecular Biology (RECOMB)*, W. Miller, M. Vingron, S. Istrail, P. Pevzner, and M. Waterman, Eds. Berlin, Germany, 2003, pp. 148–156.
[35] A. B. Kahng, I. I. Măndoiu, S. Reda, X. Xu, and A. Zelikovsky, "Design flow enhancements for DNA arrays," in *Proc. IEEE Int. Conf. Computer Design (ICCD)*, San Jose, CA, 2003, pp. 116–123.
[36] ——, "Evaluation of placement techniques for DNA probe array layout," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, CA, 2003, pp. 262–269.
[37] A. B. Kahng, I. I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky, "Scalable heuristics for design of DNA probe arrays," *J. Comput. Biol.*, vol. 11, no. 2–3, pp. 429–447, 2004.
[38] S. Kasif, Z. Weng, A. Derti, R. Beigel, and C. DeLisi, "A computational framework for optimal masking in the synthesis of oligonucleotide microarrays," *Nucleic Acids Res.*, vol. 30, no. 20, p. e106, Oct. 2002.
[39] T. Kozawa *et al.*, "Automatic placement algorithms for high packaging density VLSI," in *Proc. 20th Design Automation Conf. (DAC)*, Miami Beach, FL, 1983, pp. 175–181.
[40] F. Li and G. D. Stormo, "Selection of optimal DNA oligos for gene expression arrays," *Bioinformatics*, vol. 17, no. 11, pp. 1067–1076, Nov. 2001.
[41] R. J. Lipshutz, S. P. Fodor, T. R. Gingeras, and D. J. Lockhart, "High density synthetic oligonucleotide arrays," *Nat. Genet.*, vol. 21, no. 1, pp. 20–24, Jan. 1999.
[42] B. T. Preas and M. J. Lorenzetti, Eds. *Physical Design Automation of VLSI Systems*. Redwood City, CA: Benjamin Cummings, 1988.
[43] S. Rahmann, "Rapid large-scale oligonucleotide selection for microarrays," in *Proc. IEEE Computer Society Bioinformatics Conf. (CSB)*, Stanford, CA, 2002, pp. 54–63.
[44] ——, "The shortest common supersequence problem in a microarray production setting," *Bioinformatics*, vol. 19, Supp. 2, pp. 156–161, Sep. 2003.
[45] M. Sarrafzadeh and M. Wang, "NRG: Global and detailed placement," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, CA, 1997, pp. 532–537.
[46] R. Sengupta and M. Tompa, "Quality control in manufacturing oligo arrays: A combinatorial design approach," *J. Comput. Biol.*, vol. 9, no. 1, pp. 1–22, 2002.
[47] K. Shahookar and P. Mazumder, "VLSI cell placement techniques," *Comput. Surv.*, vol. 23, no. 2, pp. 143–220, Jun. 1991.
[48] N. A. Sherwani, *Algorithms for VLSI Physical Design Automation*. Norwell, MA: Kluwer, 1999.
[49] L. Steinberg, "The backboard wiring problem: A placement algorithm," *SIAM Rev.*, vol. 3, no. 1, pp. 37–50, Jan. 1961.
[50] A. C. Tolonen, D. F. Albeanu, J. F. Corbett, H. Handley, C. Henson, and P. Malik, "Optimized *in situ* construction of oligomers on an array surface," *Nucleic Acids Res.*, vol. 30, no. 20, p. e107, Oct. 2002.
[51] M. Wang, X. Yang, and M. Sarrafzadeh, "DRAGON2000: Standard-cell placement tool for large industry circuits," in *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, San Jose, CA, 2001, pp. 260–263.
[52] J. A. Warrington, R. Todd, and D. Wong, Eds. *Microarrays and Cancer Research*. Westboro, MA: BioTechniques Press/Eaton Pub., 2002.

Andrew B. Kahng (A'89–M'03) received the A.B. degree in applied mathematics from Harvard College, Cambridge, MA, and the M.S. and Ph.D. degrees in computer science from University of California at San Diego, La Jolla.

From 1989 to 2000, he was a member of the University of California Los Angeles computer science faculty. Since 1997, he has defined the physical design roadmap for the International Technology Roadmap for Semiconductors (ITRS), and since 2001 has chaired the U.S. and international working groups for design technology for the ITRS. He is a Professor of Computer Science and Engineering (CSE) and Electronics and Communications Engineering (ECE) at UC San Diego. He has published over 200 papers in the very-large-scale-integration (VLSI) computer-aided design (CAD) literature. His research is mainly on physical design and performance analysis of VLSI as well as the VLSI design manufacturing interface. Other research interests include combinatorial and graph algorithms and large-scale heuristic global optimization. He has been active in the MARCO Gigascale Silicon Research Center since its inception.

Prof. Kahng was the founding General Chair of the Association for Computing Machinery (ACM)/IEEE International Symposium on Physical Design, and co-founded the ACM Workshop on System-Level Interconnect Planning. He received three Best Paper awards and a National Science Foundation (NSF) Young Investigator Award.



Ion I. Măndoiu received the M.S. degree from Bucharest University, Bucharest, Romania, in 1992 and the Ph.D. degree from the Georgia Institute of Technology, Atlanta, GA, in 2000, both in computer science.

He is currently an Assistant Professor at the Computer Science and Engineering Department, University of Connecticut, Storrs. His research focuses on the design and analysis of exact and approximation algorithms for NP-hard optimization problems, particularly in the areas of very-large-scale-integration (VLSI) computer-aided design, bioinformatics, and *ad hoc* wireless networks. He has authored over 50 refereed scientific publications in these areas.

Dr. Măndoiu received the best paper award at the joint Asia-South Pacific Design Automation/VLSI Design Conference.

Sherief Reda (S'01) received the B.Sc. and M.Sc. degrees in electrical and computer engineering from Ain Shams University, Cairo, Egypt, in 1998 and 2000, respectively, and is currently working towards the Ph.D. degree at University of California at San Diego, La Jolla.

He has over 20 refereed publications in the areas of physical design, very-large-scale-integration (VLSI) test and diagnosis, combinatorial optimization, and computer-aided design (CAD) for DNA chips.

Mr. Reda received the best paper award at DATE 2002 and the first place award at the ISPD 2005 placement contest.

Xu Xu was born in Maanshan, China, in 1975. He received the B.S. degree from Special Class for Gifted Young, University of Science and Technology of China, Hefei, China, in 1998, and is currently working towards the Ph.D. degree at the Computer Science and Engineering Department, University of California at San Diego, La Jolla.

In 2002, he was with Ammocore Technology, Inc., Santa Clara, CA. In 2004, he was with Synopsys, Inc., Mountain View, CA. His research includes very large scale integration (VLSI) physical design, application of VLSI algorithms to DNA array design, and IC manufacturing cost minimization.

Alexander Z. Zelikovsky received the Ph.D. degree in computer science from the Institute of Mathematics of the Belorussian Academy of Sciences, Minsk, Belarus, in 1989.

From 1989 to 1995, he was with the Institute of Mathematics, Kishinev, Moldova. Between 1992 and 1995, he visited Bonn University and the Institut fur Informatik in Saarbrueken, Germany. He was a Research Scientist at the University of Virginia from 1995 to 1997 and a Postdoctoral Scholar at the University of California at Los Angeles (UCLA) from 1997 to 1998. In 1999, he joined the Computer Science Department, Georgia State University, Atlanta, where he is currently an Associate Professor. He is the author of more than 90 refereed publications. His research interests include very-large-scale-integration (VLSI) physical layout design, *ad hoc* wireless networks, discrete and approximation algorithms, combinatorial optimization, and computational biology.