Chapter 1

COMPUTER-AIDED OPTIMIZATION OF DNA ARRAY DESIGN AND MANUFACTURING

Andrew B. Kahng
CSE and ECE Department, University of California at San Diego
La Jolla, CA 92093-0114, USA
abk@ucsd.edu

Ion I. Măndoiu
CSE Department, University of Connecticut
371 Fairfield Rd., Unit 2155, Storrs, CT 06269-2155, USA
ion@engr.uconn.edu

Sherief Reda
CSE Department, University of California at San Diego
La Jolla, CA 92093-0114, USA
sreda@cs.ucsd.edu

Xu Xu
CSE Department, University of California at San Diego
La Jolla, CA 92093-0114, USA
xuxu@cs.ucsd.edu

Alex Z. Zelikovsky
CS Department, Georgia State University
University Plaza, Atlanta, Georgia 30303, USA
alexz@cs.gsu.edu

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. As the number and size of DNA array designs continues to grow, there is an imperative need for highly-scalable software tools with predictable solution quality to assist in the design and manufacturing process. In this chapter we review recent algorithmic and methodological advances forming the foundation for a new generation of DNA array design tools. A recurring motif behind these advances is exploiting the analogy between silicon chip design, pointing to the value of technology transfer between the 40-year old VLSI CAD field and the newer DNA array design field.

Keywords: DNA arrays, computer-aided design, design flow, border length minimization, probe placement, probe embedding, algorithms

1. 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 [45] for a survey). The number of applications is growing at an exponential rate [28], [55], already covering a diversity of fi elds 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], [25]. 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 2-dimensional pattern of synthesis sites. Probe synthesis then proceeds in successive steps, with one nucleotide (A, C, T, 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 non-masked 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 coming years with the ever-growing number of applications [28], [55], there is an urgent need for high-quality software tools to assist in the design and manufacturing process. The biggest challenges to 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 sub-micron size [2], [45] – and the increased complexity of the design process, which leads to unpredictability of design quality and design turnaround time.

In this chapter we review recent algorithmic and methodological advances addressing these challenges and already proved to yield significant solution quality and scalability improvements over existing methods. A recurring motif behind these advances is exploiting 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 DNA array design field.

The organization of the chapter is as follows. In Section 2 we introduce the main steps of the DNA array design flow. In Section 3 we formalize the synchronous and asynchronous array design problems and establish lower bounds on the achievable border length. Algorithms for the two versions of the array design problem are presented in Sections 4 and 5, respectively. In Section 6, we give empirical results comparing the presented algorithms and review novel methodologies for characterizing heuristic suboptimality scaling. Finally, we discuss enhancements of DNA array design flow in Section 7 and conclude in Section 8.

2. DNA Array Design Steps

In this section we introduce the main steps of a typical design flow for DNA arrays, noting the similarities 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 [9]. In Section 7 we will revise this flow and show how it can 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 [22], [51].

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], [44], [8], [36], [47].

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 which maximizes the minimum number of mismatches with all other sequences [44]. 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.

Deposition Sequence Design

The number of synthesis steps directly affects manufacturing time and the number of masks in the mask set, as well as the likelihood of manufacturing errors. 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 [42], [53]: Given a finite alphabet Σ (for DNA arrays $\Sigma = \{A, C, T, G\}$) and a set $P = \{p_1, ..., p_t\} \subseteq \Sigma^n$ of probes, find a minimum-length string $s_{opt} \in \Sigma^*$ such that every string of P is a subsequence of s_{opt}. (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., [10], [12], [13], [20], [26], [27], [35], [48]).

The general SCS problem is NP-hard, and cannot be approximated within a constant factor in polynomial time unless $P = NP$ [35]. 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 [26], or variations of it that add randomization, lookahead, bidirectionality, etc. (see, e.g., [42]). Some DNA array design methodologies

[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 [36].

bypass the deposition design step and use a predefined periodic deposition sequence such as $ACTGACTG\ldots$ (see, e.g., [42], [53]).

Design of Control and Test Structures

DNA array manufacturing defects can be classified as *non-catastrophic*, 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. Non-catastrophic defects are caused by systematic error sources in the VLSIPS manufacturing process, such as unintended illumination due to diffraction, internal reflection, and scattering. Their likelihood can be reduced during the physical design stage, as detailed in next section.

Catastrophic manufacturing defects affect a large fraction of the probes on the chip, and can be caused caused by missing, out-of-order, or incomplete synthesis steps, wrong or misaligned masks, etc. These defects can be detected by incorporating on the chip 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* [33]) 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 [7], [17], [49].

Physical Design

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 should not be affected by the placement of the probes on the chip or by the probe synthesis schedule. In practice, the manufacturing process is prone to synthesis errors that are highly sensitive to the actual probe placement and synthesis schedule. 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 carefully choosing manufacturing process parameters, e.g., by properly controlling exposure times and by inserting 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. 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 [34].[2]

Let $S = e_1 e_2 \dots e_K$ denote the nucleotide deposition sequence, i.e., $e_i \in \{A, C, T, G\}$ denotes the nucleotide synthesized in the ith synthesis step. Clearly, every probe in the array must be a subsequence of S. When a probe corresponds to multiple subsequences of S, one such subsequence (embedding of the probe into S) must be chosen as the schedule for synthesizing the probe. Clearly, the geometry of the masks is uniquely determined by the placement of the probes on the array and the synthesis schedule used for each probe.

More formally, the border minimization problem is equivalent to finding a *three-dimensional placement* of the probes [37, 41]: two dimensions represent the site array, and the third dimension represents the nucleotide deposition sequence S (see Figure 1.1). Each layer in the third dimension corresponds to a mask that induces deposition of a particular nucleotide (A, C, G, or T); while columns correspond to embedded probes. The border length of a given mask is computed as the number of *conflicts*, i.e., pairs of adjacent transparent and masked sites in the mask. Given two adjacent embedded probes p and p', the *conflict distance* $d(p, p')$ is the number of conflicts between the corresponding columns. The total border length of a three-dimensional placement is the sum of conflict distances between adjacent probes, and the *border minimization problem (BMP)* seeks to minimize this quantity.

We distinguish two types of DNA array synthesis. In *synchronous* synthesis, the i^{th} period $(ACGT)$ of the periodic nucleotide deposition sequence S synthesizes a single nucleotide (the i^{th}) in each probe. This corresponds to a unique (and trivially computed) embedding of each probe p in the sequence S;

[2]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.

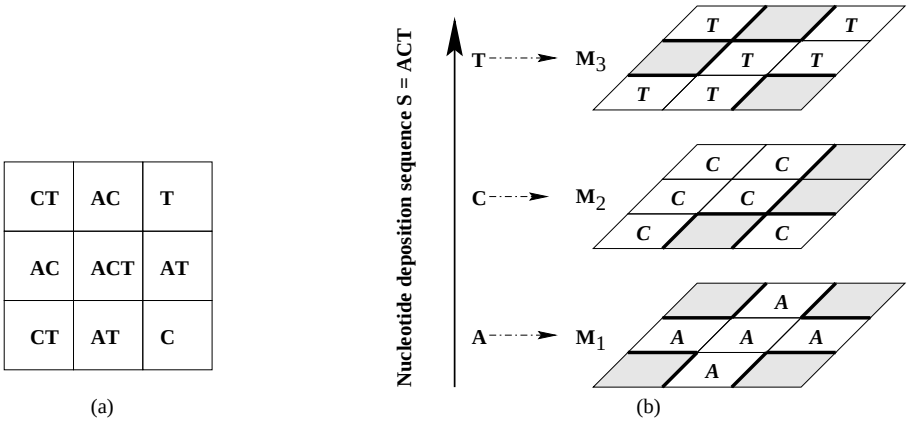

Figure 1.1. (a) Two-dimensional probe placement. (b) Three masks corresponding to nucleotide deposition sequence $S = (ACT)$. Masked sites are shaded, and borders between transparent and masked sites are thickened.

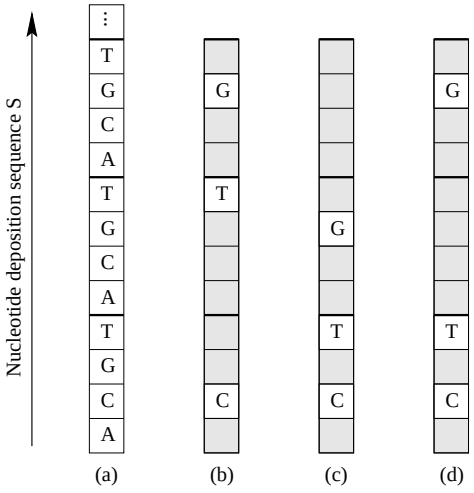

Figure 1.2. (a) Periodic nucleotide deposition sequence S. (b) Synchronous embedding of probe CTG into S; the shaded sites denote the masked sites in the corresponding masks. (c-d) Two different asynchronous embeddings of the same probe.

see Figure 1.2(a-b). On the other hand, *asynchronous* array synthesis permits arbitrary embeddings, as shown in Figure 1.2(c-d).

The border minimization problem was first considered for *uniform arrays* (i.e., arrays containing all possible probes of a given length) by Feldman and Pevzner [23], who proposed an optimal solution based on 2-dimensional Gray codes. Hannenhalli et al. [29] gave heuristics for the special case of synchronous synthesis. In this case the border-length contribution from two probes

p and p' placed next to each other (in the synchronous synthesis regime) is twice the Hamming distance between them, i.e., twice the number positions in which they differ. Hence, BMP reduces to finding a two-dimensional placement of the probes that minimizes the sum of Hamming distances between adjacent probes. The method of [29] 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 two-dimensional array of sites, using a technique similar to one previously used in VLSI design [43]. For the same synchronous context, improved probe placement algorithms were proposed in [37, 38, 40, 41]. These algorithms, drawing on techniques borrowed from the VLSI circuit placement literature, such as epitaxial growth, recursive partitioning, or window-based local re-optimization, are discussed in detail in Section 4.

The general border minimization problem, which allows *asynchronous* probe embeddings, was introduced by Kahng et al. [37], who proposed a dynamic programming algorithm that embeds a given probe optimally with respect to fixed embeddings of the probe's neighbors, and used it to decrease border length by iteratively re-embedding array probes after placing them using a synchronous placement algorithm. Asynchronous probe placement and embedding algorithms in [37] and subsequent improvements in [38, 41] are discussed in Section 4.

3. Array Design Problem Formulations and Lower Bounds

Following [37, 41], in this section we give graph-theoretical formulations and theoretical lower bounds for the synchronous and asynchronous variants of BMP.

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

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

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

The border minimization problem for synchronous array design can be cast as an optimal placement problem. In this case we let G_2 be a *two-dimensional grid graph* corresponding to the arrangement of sites in the DNA array, i.e., $V(G_2)$ has $N \times N$ vertices corresponding to array sites, and $E(G_2)$ has edge weights of 1 for every vertex pair corresponding to adjacent sites, and edge weights of 0 otherwise. Also, let H be the *Hamming graph* defined by the

set of probes, i.e., the complete graph with probes as vertices and each edge weight equal to twice the Hamming distance between corresponding probes. The border minimization problem for synchronous array design can then be formulated as follows:

Synchronous Array Design Problem (SADP). Find a minimum-cost placement of the Hamming graph H on the two-dimensional grid graph G_2.

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

Asynchronous Array Design Problem (AADP). Find embeddings into the nucleotide deposition sequence S for all given probes and a placement of the corresponding graph H' on the two-dimensional grid graph G_2 such that the cost of the placement is minimized.

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

THEOREM 1.1 *[37, 41] The total arc weight of L is a lower bound on the cost of the optimum SADP solution.*

In order to obtain non-trivial lower-bounds on the cost of the optimum AADP solution, it is necessary to establish a lower-bound on the conflict distance between two probes independent of their embedding into S. We get such a lower-bound by observing that the number of nucleotides (mask steps) common to two embedded probes cannot exceed the length of the *longest common subsequence* (LCS) of the two probes. Defi ne the *LCS distance* between probes p and p' by $lcsd(p, p') = k - |LCS(p, p')|$, where $k = |p| = |p'|$, and let L' be the directed graph over the set of probes obtained by including arcs from each probe to the 4 closest probes with respect to LCS distance, and then deleting the heaviest $4N$ arcs. Similar to Theorem 1.1, it follows that:

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

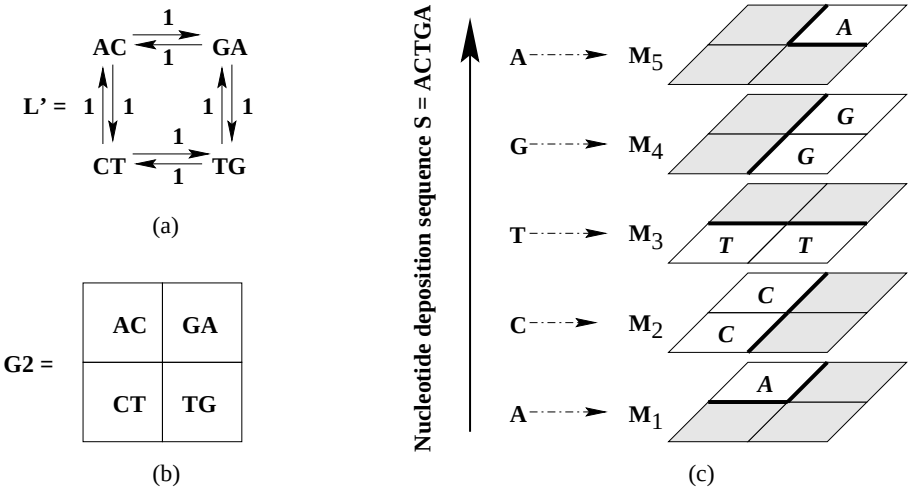

Figure 1.3. (a) Lower-bound digraph L' for the probes AC, GA, CT, and TG. The arc weight of L' is 8. (b) Optimum two-dimensional placement of the probes. (c) Optimum embedding of the probes into the nucleotide deposition supersequence $S = ACTGA$. The optimum embedding has 10 conflicts, exceeding the lower bound by 2.

THEOREM 1.2 *[37, 41] The total arc weight of L' is a lower bound on the cost of the optimum AADP solution.*

The weight of L' may be smaller than the optimum cost, since the embeddings needed to achieve LCS distance between pairs of adjacent probes may not be compatible with each other. Figure 1.3 gives one such example consisting of four dinucleotide probes, AC, GA, CT, and TG, which must be placed on a 2×2 grid. In this case, the lower bound on the number of conflicts is 8 while the optimum number of conflicts is 10.

4. Scalable Algorithms for SADP

In this section, we review recent highly-scalable heuristics for synchronous probe placement. We first describe the epitaxial growth algorithm in [37] and its highly scalable row-epitaxial version in [41]. Finally, we describe the sliding window matching heuristic for synchronous placement improvement [38] (based on optimally re-placing an independent set of probes via a reduction to minimum cost assignment), and the partition based synchronous probe placement in [40]. A recurring motif behind these algorithms is the technology transfer between the 40-year VLSI design literature and the newer field of DNA chip design.

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

---

1. Mark all grid sites as empty
2. Assign a randomly chosen probe to the center site and mark this site as full
3. While there are empty sites, do

If there exists an empty site c with all 4 neighbors full, then

Find probe $p(c) \in P$ with minimum sum of Hamming distances to the neighboring probes

Assign probe $p(c)$ to site c and mark c as full

Else

For each empty site c with $i > 0$ adjacent full sites, find probe $p(c) \in P$ with minimum sum S of Hamming distances to the probes in full neighbors, and let $norm\_cost(c) = k_i S/i$.

Let c^* be the site with minimum $norm\_cost$

Assign probe $p(c^*)$ to site c^* and mark c^* as full

Figure 1.4. The Epitaxial Algorithm

Epitaxial Growth SADP Algorithms

In this section, we describe the so-called *epitaxial growth* approach to SADP and discuss some effi cient implementation details [37, 41]. Epitaxial, or seeded crystal growth, placement is a technique that has been well-explored in the VLSI circuit placement literature [46, 50]. The technique essentially grows a two-dimensional placement around a single starting seed.

The algorithm in [29] , which fi nds a TSP tour and then threads it into the array, optimizes directly only half of the pairs of adjacent probes in the array (those corresponding to tour edges). Intuitively, the epitaxial algorithm (see Figure 1.4) attempts to make full use of the available information during placement. The algorithm places a random probe at the center of the array, and then iteratively places probes in sites adjacent to already-placed probes so as to greedily minimize the average number of conflicts induced between *all* newly created pairs of neighbors. Sites with more fi lled neighbors have higher priority to be fi lled; in particular, sites with 4 known neighbors have the highest priority. To avoid repeated distance computations, the algorithm maintains for each border site a list of probes sorted by normalized cost. For each array site, this list is computed at most four (and on the average two) times, i.e., when one of the neighboring sites is being fi lled while the site is still empty.

While the epitaxial algorithm achieves good results, it runtime becomes impractical for DNA chips with dimensions of 300×300 or more. Any synchronous placement method can be trivially scaled by partitioning the set of probes and the probe array into K subsets ("chunks"), then solving K inde-

pendent placement problems. While this ensures linear scaling of runtime, 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. In [41] it is noted that better solution quality is achieved for a different scalable variant of the epitaxial algorithm called the *row-epitaxial* algorithm. There are three main distinguishing features of the row-epitaxial variant:

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

(2) The sites are filled with crystallized probes in a predefined order, namely, row by row and within a row from left to right;

(3) The probe filling each site is chosen as the best candidate not among *all* remaining ones, but among a bounded number of them (the not yet "crystallized" probes within the next k_0 rows, where k_0 is a parameter of the algorithm).

Feature (1) is critical for compensating the loss in solution quality due to the reduced search space imposed by (2) and (3). Since the initial placement must be very fast to compute, one cannot afford using any two-dimensional placement based on computing all pairwise distances between probes (such as TSP-based placement in [29]). Possible initial placement algorithms can be based on space-filling curve (e.g., Gray code) ordering [11]; indeed such orderings have had success in the VLSI context [5]. As noted in [41], an excellent initial placements is obtained by simply ordering the probes lexicographically (this can be done in linear time by radix sort) and then threading them as in [29]. Features (2) and (3) speed-up the algorithm significantly, with the number k_0 of look-ahead rows allowing a fine tradeoff between solution quality and runtime.

Highly Scalable Algorithms for Synchronous Placement Improvement

In the early VLSI placement literature, iterative placement improvement methods relied on weak neighborhood operators such as pair-swap, leveraged by meta-heuristics such as simulated annealing. More recently, strong neighborhood operators have been proposed which improve larger portions of the placement. For example, the DOMINO approach [21] iteratively determines an optimal reassignment of all objects within a given window of the placement. The end-case placer of [14] 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 [21] around the placement; cf. cycling and overlapping [32], row-ironing [14], 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. However, randomly chosen pairs of probes are extremely unlikely to be swappable with reduction in border cost. On the other hand, optimal probe re-placement of an entire window of probes is not practical even for very small window sizes. However, as noted in [38], optimal probe re-placement of large sets of *independent* (i.e., non-adjacent) probes reduces to computing a minimum cost assignment, where the cost of assigning a probe p to a cell c is given by the sum of Hamming distances between p and the probes placed in the four cells adjacent to c. For a set of t independent cells, computing the minimum cost assignment requires $O(t^3)$ time. Full-chip application with practical runtime is achieved by iteratively choosing the independent set from a sliding window that is moved around the array; this approach is a reminiscence of early work on electronic circuit placement by [3, 52].

Following extensive algorithm engineering, the following implementation of the sliding window method was found to work best [38]. (1) First, radix-sort all probes lexicographically and then perform 1-threading as in [29]. (2) For each sliding $W_0 \times W_0$ window, choose one random maximal independent set of sites and determine the cost of (asynchronous) reassignment of each associated probe to each site, then reassign probes according to the minimum weight perfect matching in the resulting weighted bipartite graph. (3) The window slides in rows, beginning in the top-left corner of the array; at each step, it slides horizontally to the right as far as possible while maintaining a prescribed amount of *window overlap*. After the right side of the array is reached, the window returns to the left end of the next row while maintaining the prescribed overlap with the preceding row. When the bottom side of the array is reached, the window returns to the top-left corner. The experiments in [38] have shown that an overlap equal to half the window size gives best results. (4) The window-sliding continues until an entire pass through the array results in less than 0.1% reduction of border cost. Figure 1.5 illustrates the heuristic tuning with respect to varying window sizes.

Partition Based Probe Placement

Recursive partitioning has been the basis of numerous successful VLSI placement algorithms [6], [15], [54] since it produces placements with acceptable wirelength 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 hyper-edge cut between these sets. The min-cut goal is typically achieved through the use of the Fiduccia-Mattheyses procedure [24], often in a multilevel framework [15].

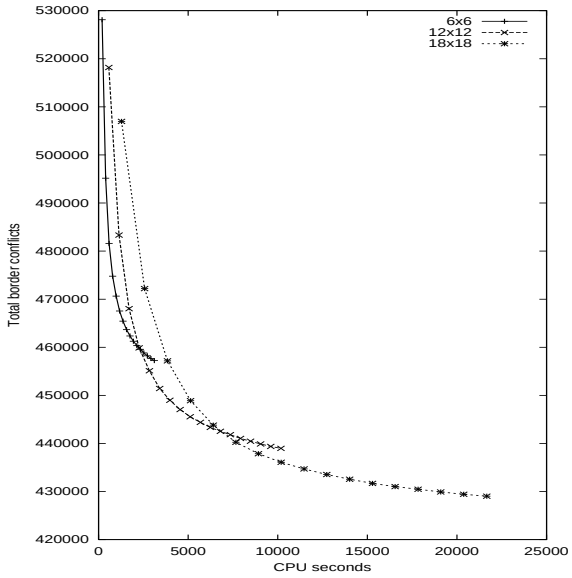

Figure 1.5. Solution quality vs. runtime for Synchronous Sliding-Window Matching with varying window size; array size = 100×100.

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 which finalizes probe placements at the border between the two partitions. In this section we describe the *centroid-based quadrisection* method proposed in [40], which 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 equally sized subsets R_1, R_2, R_3 and R_4. The probes in R are partitioned among R_i's by picking a *representative*, or *centroid*, probe C_i for each R_i, and then iteratively assigning remaining probes to the subset R_i whose representative is closest in Hamming distance. The procedure for selecting the four centroids, reminiscent of the k-center approach to clustering studied by Alpert et al. [4] and of methods used in large-scale document classification [19], is described in Figure 1.6.

The complete partitioning-based placement algorithm for DNA arrays is given in Figure 1.7. The algorithm recursively quadrisects every partition at a given level, assigning the probes so as to minimize distance to the centroids of

Input: Partition (set of probes) R
Output: Probes C_0, C_1, C_2, C_3 to be used as centroids for the 4 sub-partitions

---

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)

Figure 1.6. The **SelectCentroid()** procedure for selecting the centroid probes of sub-partitions.

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

Figure 1.7. Partitioning-based DNA probe placement heuristic.

sub-partitions.[4] A multi-start heuristic in the innermost of the three nested *for* loops of Figure 1.7, whereby r different random probes are tried as seed C_0, and the result that minimizes the total distance to the centroids is selected. 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.[5]

Once the maximum level L of the recursive partitioning is attained, detailed placement is executed via a modified version of the row-epitaxial algorithm. Since the basic implementation of the row-epitaxial algorithm treats the last locations within a region "unfairly" (e.g., only one candidate probe will remain available for placing in a region's last location), the modified algorithm permits "borrowing" probes from the next region. The modified row-epitaxial algorithm is also "border-aware", that is, it takes into account Hamming distances to the already placed probes in adjacent regions.

5. In-Place Optimization of Probe Embeddings

Experiments in [37, 38] indicate that separate optimization of probe placement and embedding yields better results for AADP than simultaneous optimization of the two degrees of freedom. For example, the asynchronous version of the epitaxial algorithm [37] and the asynchronous version of sliding-window matching [38] are both dominated by algorithms implementing the following two-step flow:

Step (i). Find a two-dimensional placement based on synchronous embedding for the probes (using, e.g., the row-epitaxial and sliding-window matching algorithms discussed in the previous section, or the TSP+1-Threading of [29]).

Step (ii). Iteratively optimize probe embeddings, *without changing their location on the array.*

In this section we consider the second step of the above flow. We first present a dynamic programming algorithm for optimally embedding a single probe with respect to its neighbors, as well as a lower-bound on the optimum border cost for in-place probe embedding [37]. We then present three methods

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

[5]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 $C_i[t] = \bigcup_s \frac{N_{s,t}}{N_i} \cdot s$, 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 t-th position. The Hamming distance between a probe p and C_i is $d(p, C_i) = \frac{1}{N_i} \sum_t \sum_{s \neq p[t]} N_{s,t}$.

Input: Nucleotide deposition sequence $S = s_1 s_2 \ldots s_K$, $s_i \in \{A, C, G, T\}$; set X of probes already embedded into S; and unembedded probe $p = p_1 p_2 \ldots p_k$, $p_i \in \{A, C, G, T\}$

Output: The minimum number of conflicts between an embedding of p and probes in X, along with a minimum-conflict embedding

---

1. For each $j = 1, \ldots, K$, let x_j be the number of probes in X which have a non-blank letter in j^{th} position.

2. $cost(0, 0) = 0$; For $i = 1, \ldots, k$, $cost(i, 0) = \infty$

3. For $j = 1, \ldots, K$ do

$cost(0, j) = cost(0, j-1) + x_j$

For $i = 1, \ldots, k$ do

If $p_i = s_j$ then
$cost(i, j) = \min\{cost(i, j-1) + x_j,\ cost(i-1, j-1) + |X| - x_j\}$
Else $cost(i, j) = cost(i, j-1) + x_j$

4. Return $cost(k, K)$ and the corresponding embedding of s

Figure 1.8. The Single Probe Alignment Algorithm

for iterative in-place probe embedding optimization [37, 41], and conclude with a useful theoretical bound on the amount of improvement available during this optimization step.

Optimum Embedding of a Single Probe

The basic operation used by in-place embedding optimization algorithms is to find the optimum embedding of a probe when the adjacent sites contain already embedded probes. In other words, the goal is to simultaneously align the given probe s to its embedded neighboring probes, while making sure this alignment gives a feasible embedding of s in the nucleotide deposition sequence S. In this section we present an efficient dynamic programming algorithm given in [37] for computing this optimum alignment.

The Single Probe Alignment algorithm (see Figure 1.8) essentially computes a shortest path in a specific directed acyclic graph $G = (V, E)$. Let p be the probe to be aligned, and let X be the set of already embedded probes adjacent to p. Each embedded probe $q \in X$ is a sequence of length $K = |S|$ over the alphabet $\{A, C, G, T, b\}$, with the j^{th} letter of q being either a blank or s_j, the j^{th} letter of the nucleotide deposition sequence S. The graph G (see Figure 1.9) has vertex set $V = \{0, \ldots, k\} \times \{0, \ldots, K\}$ (where k is the length of the probe p and K is the length of the deposition sequence S), and edge set $E = E_{horiz} \cup E_{diag}$ where

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

and

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

The cost of a horizontal edge $(i, j-1) \rightarrow (i, j)$ is defined as the number of embedded probes in X which have a non-blank letter on j^{th} position, while the cost of a diagonal edge $(i-1, j-1) \rightarrow (i, j)$ is equal to the number of embedded probes of X with a blank on the j^{th} position. The Single Probe Alignment algorithm computes the shortest path from the source node $(0, 0)$ to the sink node (k, K) using a topological traversal of G, which corresponds to the optimum embedding of s:

THEOREM 1.3 *The algorithm in Figure 1.8 returns, in $O(kK)$ time, the minimum number of conflicts between an embedding of s and the adjacent embedded probes X (along with a minimum-conflict embedding of s).*

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

Remarks. The above dynamic programming algorithm can be easily extended to find the optimal simultaneous embedding of $n > 1$ probes. The corresponding directed acyclic graph G consist of $k^n K$ nodes $(i_1, \ldots, i_n, j)$, where $0 \leq i_l \leq k$, $1 \leq j \leq K$. All arcs into $(i_1, \ldots, i_n, j)$ come from nodes $(i'_1, \ldots, i'_n, j-1)$, where $i'_l \in \{i_l, i_l - 1\}$. Therefore the in-degree of each node is at most 2^n. The weight of each edge is defined as above such that each finite weight path defines embeddings for all probes and the weight equals the number of conflicts. Finally, computing the shortest path between $(0, \ldots, 0)$ and $(k, \ldots, k, K)$ can be done in $O(2^n k^n K)$ time. The probe alignment algorithm can also be extended to handle practical concerns such as pre-placed control probes, presence of polymorphic probes, unintended illumination between non-adjacent array sites, and position-dependent border conflict weights, we refer the reader to [38, 41] for details.

Algorithms for Iterative In-Place Embedding Optimization

Batched Greedy [37]. A natural greedy algorithm is to find a probe that offers largest cost gain from optimum re-embedding with respect to the (fixed) embeddings of its neighbors, perform this re-embedding, and repeat these steps until no further improvement is possible. The *batched* version of the greedy

algorithm (see Figure 1.10) trades some gain in re-embedding steps for faster runtime. During each *batched* phase the algorithm attempts to re-embed all probes in the order given by cost gains at the beginning of the phase. The algorithm gains in effi ciency by avoiding to update probe gains after each probe re-embedding.

Chessboard Optimization [37]. The main idea behind our so-called "Chessboard" algorithm is to maximize the number of *independent* re-embeddings, where two probes are independent if changing the embedding of one does not affect the optimum embedding of the other. It is easy to see that if we bicolor our grid as we would a chessboard, then all white (resp. black) sites will be independent and can therefore be simultaneously, and optimally, re-embedded. The Chessboard Algorithm (see Figure 1.11) alternates re-embeddings of black and white sites until no improvement is obtained.

A 2×1 version of the Chessboard algorithm partitions the array into iso-oriented 2×1 tiles and bicolors them. Then using the multi-probe alignment algorithm (see the remark in Section 5.0) with $n = 2$ it alternatively optimizes the black and white 2×1 tiles.

Sequential Probe Re-Embedding [41]. In this method, probes are sequentially re-embedded optimally with respect to their neighbors in a row-by-row order. A shortcoming of the Batched Greedy and Chessboard algorithms is that, by always re-embedding *independent* sets of probes, it takes longer to propagate the effects of a new embedding. Performing the re-embedding sequentially permits faster propagation and convergence to a better local optimum.

A Lower Bound for In-Place Probe Re-Embedding

Let LG be a grid graph with weights on edges equal to the LCS distance between endpoint probes. The following theorem gives a lower bound that is very useful in assesing the quality of in-place probe re-embedding algorithms:

THEOREM 1.4 *[37] The total edge weight of the graph LG is a lower bound on the optimum AADP solution cost with a given placement.*

6. Empirical Results

In this section we give experimental results comparing algorithms introduced in previous sections. Unless otherwise specifi ed, experiments reported in this chapter were performed on test cases obtained by generating each probe candidate uniformly at random and reported numbers are averages over 10 random instances. The probe length was set to 25, which is the typical value for commercial arrays [1]. We used the canonical periodic deposition sequence, $(ACTG)^{25}$. All reported runtimes are for a 2.4 GHz Intel Xeon server with 2GB of RAM running under Linux.

Chip	LB	TSP+1Thr		Row-Epitaxial		SWM		SA	
Size	Cost	Cost Gap	CPU	Cost Gap	CPU	Cost Gap	CPU	Cost Gap	CPU
100	0.41M	0.55M 35.3	113	0.50M 22.5	108	0.60M 47.7	2	0.58M 42.4	20769
200	1.51M	2.14M 41.6	1901	1.91M 26.6	1151	2.36M 56.1	8	2.42M 59.9	55658
300	3.23M	4.67M 44.3	12028	4.18M 29.4	3671	5.19M 60.6	19	5.50M 70.2	103668
500	8.46M	12.70M 50.1	109648	11.18M 32.2	10630	13.75M 62.5	50	15.43M 82.5	212390

Table 1.1. Total border cost, gap from the synchronous placement lower-bound (in percents), and CPU time (in seconds) for the TSP threading (TSP+1Thr), the row-epitaxial (Row-Epitaxial), and sliding-window matching (SWM) heuristics, and the simulated annealing algorithm (SA).

In a first set of experiments we compare synchronous probe placement heuristics: the TSP 1-threading heuristic of [29] (TSP+1Thr), the Row-Epitaxial and sliding-window matching (SWM) heuristics of [38], a simulated annealing algorithm (SA), and the partitioning based algorithm. These experiments use 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 6×6 window over the chip in the same way as the SWM algorithm (with overlap 3). For every window position, SA picks 2 random probes in the window and swaps them with probability 1 if the swap improves 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. A number of 6^3 SA iterations was performed for every window position.

Table 1.1 shows that among TSP+1Thr, Row-Epitaxial, SWM, and SA heuristics, Row-Epitaxial is the algorithm with 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.

Table 1.2 gives results for the recursive partitioning algorithm (RPART) with recursion depth L varying between 1 and 3. Compared to Row-Epitaxial, recursive partitioning based placement achieves improved runtime and similar or better solution quality.

In a second set of experiments we compared the three probe embedding algorithms in Section 5 on random instances with chip sizes between 100 and 500 and an initial two-dimensional placement obtained using TSP+1-threading. All algorithms were stopped when the improvement cost achieved in one iteration over the whole chip drops below 0.1% of the total cost. The results in Table 1.3 show that sequential re-embedding of the probes in a row-by-row order yields the smallest border cost with a runtime similar to that of the other methods.

In another series of experiments, we ran complete placement and embedding flows obtained by combining each of the five two-dimensional placement algo-

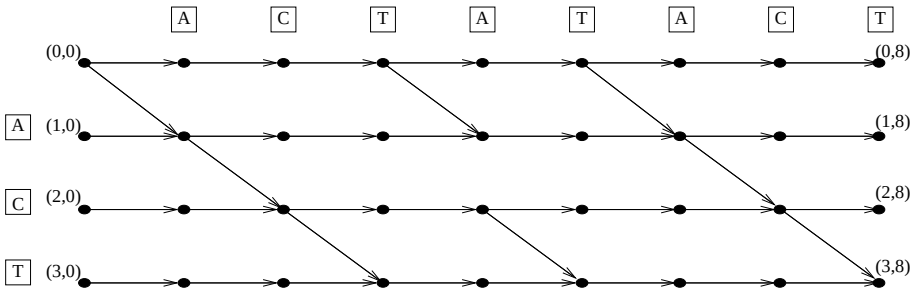

Figure 1.9. Directed acyclic graph G_1 representing possible embeddings of probe $p = ACT$ into the nucleotide deposition sequence $S = ACTATACT$.

Chip Size	LB Cost	RPART $L = 1$			RPART $L = 2$			RPART $L = 3$		
		Cost	Gap	CPU	Cost	Gap	CPU	Cost	Gap	CPU
100	0.41M	0.48M	16.1	69	0.49M	20.0	24	0.50M	23.1	10
200	1.51M	1.81M	19.9	992	1.87M	23.4	283	1.92M	27.2	81
300	3.23M	4.14M	27.9	3529	4.07M	26.0	1527	4.18M	29.1	240
500	8.46M	11.28M	33.4	10591	11.05M	30.6	9678	11.13M	31.6	3321

Table 1.2. Total border cost, gap from the synchronous placement lower-bound (in percents), and CPU time (in seconds) for the recursive partitioning algorithm with $r = 10$ and recursion depth L varying between 1 and 3.

Chip Size	LB Cost	Batched Greedy			Chessboard			Sequential		
		Cost	Gap	CPU	Cost	Gap	CPU	Cost	Gap	CPU
100	0.36M	0.45M	25.7	40	0.44M	20.5	54	0.44M	19.9	64
200	1.43M	1.80M	26.3	154	1.72M	20.9	221	1.72M	20.3	266
300	3.13M	3.97M	26.7	357	3.80M	21.5	522	3.77M	20.6	577
500	8.59M	10.92M	27.1	943	10.43M	21.4	1423	10.38M	20.9	1535

Table 1.3. Total border cost, gap from the synchronous placement lower-bound (in percents), and CPU time (in seconds) for the batched greedy, chessboard, and sequential in-place re-embedding algorithms.

Input: Feasible AADP solution, i.e., placement in $G2$ of probes embedded in S
Output: A heuristic low-cost feasible AADP solution

While there exist probes which can be re-embedded with gain in cost do

  Compute gain of the optimum re-embedding of each probe.

  Unmark all probes

  For each unmarked probe p, in descending order of gain, do

    Re-embed p optimally with respect to its four neighbors

    Mark p and all probes in adjacent sites

Figure 1.10. The Batched Greedy Algorithm

Input: Feasible AADP solution, i.e., placement in $G2$ of probes embedded in S
Output: A heuristic low-cost feasible AADP solution

Repeat until there is no gain in cost

  For each site (i, j), $1 \leq i, j \leq N$ with $i + j$ even, re-embed probe optimally with respect to its four neighbors

  For each site (i, j), $1 \leq i, j \leq N$ with $i + j$ odd, re-embed probe optimally with respect to its four neighbors

Figure 1.11. The Chessboard Algorithm

Chip	LB	TSP+1Thr			Row-Epitaxial			SWM			SA		
Size	Cost	Cost	Gap	CPU	Cost	Gap	CPU	Cost	Gap	CPU	Cost	Gap	CPU
100	0.22M	0.44M	99.5	113	0.42M	88.3	161	0.44M	99.8	93	0.46M	107.6	11713
200	0.80M	1.72M	115.8	1901	1.61M	101.4	1368	1.72M	115.6	380	1.84M	130.9	42679
300	—	3.80M	—	12028	3.53M	—	3861	3.80M	—	861	4.16M	—	101253
500	—	10.43M	—	109648	9.46M	—	12044	10.16M	—	2239	11.57M	—	222376

Table 1.4. Total border cost, gap from the synchronous placement lower-bound (in percents), and CPU time (in seconds) for the TSP threading (TSP+1Thr), the row-epitaxial (Row-Epitaxial), and sliding-window matching (SWM) heuristics, and the simulated annealing algorithm (SA) followed by sequential in-place probe re-embedding.

rithms compared above with the sequential in-place re-embedding algorithm. Results are given in Tables 1.4-1.5. 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 other flows with similar or lower runtime.

Chip Size	LB Cost	RPART $L=1$ Cost	Gap	CPU	RPART $L=2$ Cost	Gap	CPU	RPART $L=3$ Cost	Gap	CPU
100	0.22M	0.39M	78.3	123	0.40M	81.1	44	0.41M	86.2	10
200	0.80M	1.52M	90.9	1204	1.55M	93.5	365	1.57M	97.0	101
300	—	3.49M	—	3742	3.41M	—	1951	3.43M	—	527
500	—	9.55M	—	11236	9.36M	—	10417	9.30M	—	3689

Table 1.5. Total border cost, gap from the synchronous placement lower-bound (in percents), and CPU time (in seconds) for the recursive partitioning algorithm with $r = 10$ and recursion depth L varying between 1 and 3, followed by sequential in-place probe re-embedding.

Quantified Suboptimality of Placement Algorithms

As noted in the introduction, next-generation of DNA probe arrays will contain up to one hundred million probes, orders of magnitude more than current designs. Thus, it is of interest to study not only runtime scaling for available heuristics, but also the scaling of their suboptimality. Following [40], we present next an experimental framework for quantifying suboptimality of probe placement heuristics This framework, inspired by similar studies in the area of VLSI placement [30],[16],[18], comprises two basic types of instance scaling.

- **Instances with known optimum solution.** These instances consist of all 4^k probes of length k padded with the same prefi x up to the prescribed probe length. These instances are are placeable such that every probe has border cost of 2 to each of its neighboring probes using 2-dimensional Gray codes [23] (see Figure 1.12).

- **Instances with known suboptimal solutions.** Because constructed instances with known optimum solutions are not representative of "real" instances, we also apply a technique of [30] that allows real instances to be scaled, such that they offer insights into 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 4 times the optimum border cost for I, plus the border cost between the copies of I; see Figure 1.13. If a heuristic H places I' with cost $c_H(I') \geq 4 \cdot c_H(I)$, then we may infer

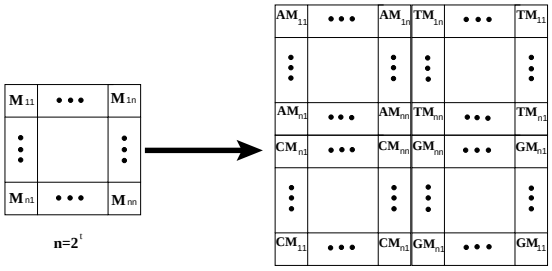

Figure 1.12. 2-dimensional Gray code placement.

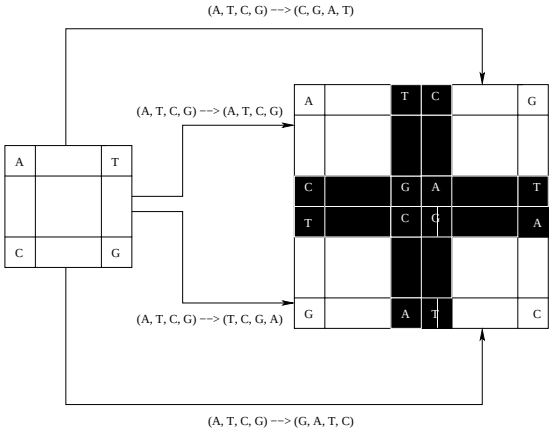

Figure 1.13. Scaling construction used in the suboptimality experiment.

that the heuristic's suboptimality is growing by at least a factor $\frac{c_H(I')}{4 \cdot c_H(I)}$. On the other hand, if $c_H(I') < 4 \cdot c_H(I)$, then the heuristic's solution quality would be said to scale well on this class of instances.

Table 1.6 shows results from executing the various placement heuristics on instances with known optimum solution. We see from these results that sliding-window matching 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 [16] 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 1.7 shows the results obtained by running the synchronous placement heuristics on scaled versions of random DNA probe sets, with original instances

Chip Size	Optimal Cost	TSP+1Thr Cost	TSP+1Thr Gap	Row-Epitaxial Cost	Row-Epitaxial Gap	SWM Cost	SWM Gap	RPART Cost	RPART 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 1.6. Comparison of placement algorithms performance on instances with known optimal solution. SW matching is using a window size of 20 x 20 and an overlap of 10. Row-epitaxial uses $10,000/chipsize$ lookahead rows.

Original Size	Row Epitaxial U-Bound	Row Epitaxial Actual	Row Epitaxial Ratio	SWM U-Bound	SWM Actual	SWM Ratio	RPART U-Bound	RPART Actual	RPART 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

Table 1.7. Suboptimality of placement algorithm performance on scaled benchmarks. SW matching is using a window size of 20 x 20 and a step of 10. Row epitaxial uses $10,000/chipsize$ lookahead rows.

ranging in size from 100 x 100 to 500 x 500. The results show that in general, 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.

7. Flow Enhancements

As noted in [39], the basic DNA array design flow described in Section 2 can be signifi cantly 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 Figure 1.14 by the dashed arcs, are similar to flow enhancements that have proved very effective in the VLSI design context [22], [51]. In this section we describe two such enhancements, both aiming at further reductions in total border length. The fi rst enhancement is a tighter integration between probe

placement and embedding. 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. These enhancements yield significant improvements (up to 15%) in border length compared to best flows in [38, 41].

Problem Formulation for Integrated Probe Selection and Physical Design

To integrate probe selection and physical design, one can pass the entire pools of candidates for each probe to the physical design step (Figure 1.14). As discussed in Section 2, candidate probes 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 [39]:[6]

Integrated DNA Array Design Problem

Given:

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

Find:

1. Probes $p_{ij} \in P_i$ for every $i = 1, \ldots, N^2$,
2. A deposition sequence $S = s_1, \ldots, s_K$ which is a supersequence of all selected probes p_{ij},

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

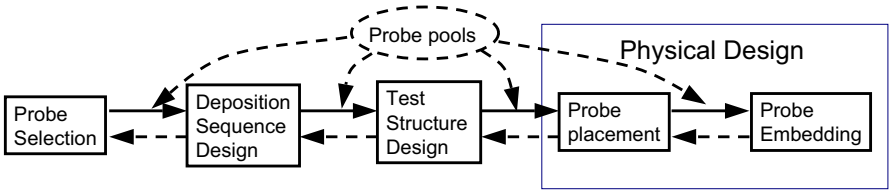

Figure 1.14. A typical DNA array design flow with solid arcs and proposed enhancements represented by dashed arcs.

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:

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

Although the flow in Figure 1.14 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 and will be used as building blocks for constructing integrated optimization flows. They are "probe pool" versions of the Row-epitaxial and re-embedding algorithms described in previous sections, 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. 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 on the 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 $\{A, C, T, G, Blank\}$, where A, C, T, G denote embedded nucleotides and $Blank$'s 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; (3) Traversing array cells, again in row-by-row order, and placing at each location the best probe candidate – i.e., the candidate having the minimum number of conflicts with already placed neighbors – 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 5. 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 array cell and for each candidate p_{ij} from the associated pool of probe candidates, an embedding having minimum number of conflicts with the existing embeddings of the neighbors is computed, and then the best embedded candidate probe is used to replace the current one.

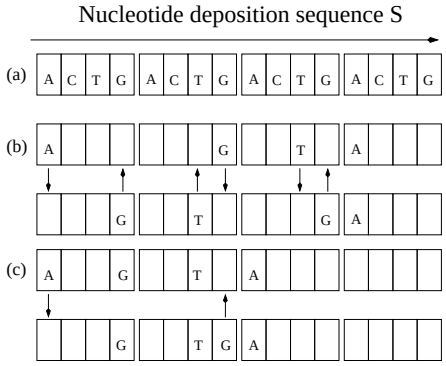

Figure 1.15. (a) Periodic deposition sequence. (b) Synchronous embedding of the probes $AGTA$ and $GTGA$ gives 6 border conflicts (indicated by arrows). (c) "As soon as possible" asynchronous embedding of the probes $AGTA$ and $GTGA$ gives only 2 border conflicts.

Improved Integration of Probe Placement and Embedding

As noted in [37], allowing asynchronous embeddings leads to further reductions in border length compared to synchronous embedding (e.g., contrast (b) and (c) in Figure 1.15). An interesting question is finding the best order in which the placement and embedding degrees of freedom should be exploited. Existing methods [37, 38, 41] can be divided into two main 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 [38, 41]) outperform the methods in the first class (e.g., the asynchronous epitaxial algorithm in [37]) in terms of both runtime and solution quality.

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 3 steps:

- Synchronous embedding of the probes.
- Probe placement with costs given by the Hamming distance between the synchronous probe embeddings.
- Iterated sequential probe re-embedding.

In [39] we noted that significant reductions in border cost are possible by performing the placement based on asynchronous, rather than synchronous, embeddings of the probes, and therefore proposed a modified scheme as follows:

- Asynchronous embedding of the probes.

- Placement with costs given by the Hamming distance between the fixed asynchronous probe embeddings.
- 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 implementation proposed in [39] the first step consists of embedding each probe using an "as soon as possible," or *ASAP*, synthesis schedule (see Figure 1.15(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, p' picked from the uniform distribution. When performing synchronous embedding, the length of the deposition sequence is $4 \times 25 = 100$. The probability that any one of the 100 synthesis step 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.

We compared ASAP and synchronous initial embeddings on test cases ranging in size from 100×100 to 500×500. For both embedding strategies, the second and third steps are implemented using REPTX and sequential in-place probe re-embedding, respectively. Tables 1.8 and 1.9 give the border-length and CPU time (in seconds) after each flow step. Remarkably, 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

Chip	Synchronous Initial Embedding			ASAP Initial Embedding			%
Size	Sync.	REPTX	Sequential	ASAP	REPTX	Sequential	Impr.
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 1.8. Total border cost (averages over 10 random instances) for synchronous and ASAP initial probe embedding followed by row-epitaxial and sequential in-place probe re-embedding.

Chip	Synchronous Initial Embedding			ASAP Initial Embedding		
Size	Sync+REPTX	Sequential	Total	ASAP+REPTX	Sequential	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

Table 1.9. CPU seconds (averages over 10 random instances) for synchronous and ASAP initial probe embedding followed by row-epitaxial and sequential in-place probe re-embedding.

converge to a locally optimal solution (the number of iterations drops from 9 to 3 on the average).

Integrated Probe Selection and Physical Design

Two different methods for exploiting the availability of multiple probe candidates during placement and embedding were proposed in [39]. A first method uses the pool versions of the row-epitaxial and sequential in-place probe re-embedding algorithms described above. 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) using border costs given by the Hamming distance between the ASAP embeddings.
- Run the pool version of the sequential in-place re-embedding algorithm.

The second method preserves the separation between candidate selection and placement+embedding. However, probe selection is modified 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, the modified probe selection algorithm picks 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 in 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 the ASAP embeddings.
- Run the iterated sequential in-place probe re-embedding algorithm, again using only selected candidates.

Table 1.10 gives the border-length and the runtime (in CPU seconds) for the two methods (each number represents the average over 10 test cases of the given size). The pool version of the recursive-partitioning uses $L = 3$. In these experiments, the number of candidates available for each probe is varied 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 non-uniformly with the method and chip size. For small chips the first method gives better 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 8 probe candidates). The second method is faster than 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.

8. Conclusions

In this chapter we have reviewed several recent algorithmic and methodological advance in DNA array design, focusing on minimizing the total mask border length during probe placement and embedding. Unlike VLSI placement, where placer suboptimality generally increases with instance size, empirical experimental results suggest that the opposite trend holds 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

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	0.39M	217	–	0.38M	115	–	0.39M	217	–	0.38M	115	–
	2	0.37M	1040	4.3	0.37M	676	0.9	0.38M	212	3.2	0.36M	114	3.6
	4	0.36M	1796	8.2	0.36M	1274	5.0	0.36M	193	6.6	0.35M	127	7.0
	8	0.34M	3645	11.8	0.34M	2605	8.7	0.35M	191	9.8	0.34M	109	9.4
	16	0.33M	7315	15.2	0.33M	5003	12.2	0.34M	185	12.8	0.33M	121	11.6
200	1	1.48M	1416	–	1.45M	1012	–	1.48M	1416	–	1.45M	1012	
	2	1.44M	6278	3.1	1.44M	7281	0.6	1.44M	1176	3.3	1.41M	946	2.5
	4	1.39M	12750	6.6	1.39M	13231	4.1	1.39M	1189	6.6	1.36M	932	5.9
	8	1.33M	27382	10.1	1.33M	26413	7.7	1.34M	1121	9.9	1.31M	957	9.2
	16	1.28M	44460	13.5	1.28M	52400	11.2	1.29M	1117	13.1	1.28M	971	11.7
300	1	3.25M	2971	–	3.22M	2975	–	3.25M	2971	–	3.22M	2975	–
	2	3.19M	14956	1.9	3.18M	13161	1.1	3.14M	2724	3.2	3.05M	2134	5.4
	4	3.09M	26514	4.7	3.09M	24671	3.9	3.02M	2771	7.0	2.94M	2118	8.8
	8	2.99M	51226	8.0	2.99M	45607	7.3	2.92M	2603	10.0	2.83M	2079	12.0
	16	2.88M	98189	11.3	2.88M	85311	10.6	2.84M	2760	12.6	2.71M	2247	15.9
500	1	8.69M	6842	–	8.65M	5608	–	8.69M	6842	–	8.65M	5608	–
	2	8.61M	51847	0.9	8.61M	41409	0.4	8.41M	6090	3.2	8.27M	5468	4.3
	4	8.48M	86395	2.4	8.48M	94566	1.9	8.11M	6709	6.7	7.96M	5591	8.0
	8	8.25M	161651	5.1	8.25M	213264	4.6	7.81M	6085	10.1	7.64M	5782	11.7
	16	–	–	–	–	–	–	7.52M	5986	13.5	7.45M	5601	13.9

Table 1.10. Total border cost and runtime (averages over 10 random instances) for the two methods of combining probe placement and embedding with probe selection. The improvement (in percents) is relative to the single-candidate version of the respective method.

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. Other direction of future research is to find formulations and algorithms 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] http://www.affymetrix.com

[2] http://www.perlegen.com

[3] S. Akers, "On the Use of the Linear Assignment Algorithm in Module Placement," *Proc. 1981 ACM/IEEE Design Automation Conference (DAC'81)*, pp. 137–144.

[4] C. J. Alpert and A. B. Kahng, "Geometric Embeddings for Faster (and Better) Multi-Way Netlist Partitioning" *Proc. ACM/IEEE Design Automation*

Conf., 1993, pp. 743-748.

[5] C. J. Alpert and A. B. Kahng, "Multi-Way Partitioning Via Spacefi lling Curves and Dynamic Programming," *Proc. 1994 ACM/IEEE Design Automation Conference (DAC'94)*, pp. 652-657.

[6] C.J. Alpert and A.B. Kahng, "Recent directions in netlist partitioning: A survey", *Integration: The VLSI Jour.* 19 (1995), pp. 1-81.

[7] N. Alon, C. J. Colbourn, A. C. H. Lingi and M. Tompa, "Equireplicate Balanced Binary Codes for Oligo Arrays", *SIAM Journal on Discrete Mathematics* 14(4) (2001), pp. 481-497.

[8] A.A. Antipova, P. Tamayo and T.R. Golub, " A strategy for oligonucleotide microarray probe reduction", *Genome Biology* 2002 3(12):research0073.1-0073.4

[9] M. Atlas, N. Hundewale, L. Perelygina and A. Zelikovsky, "Consolidating Software Tools for DNA Microarray Design and Manufacturing", *Proc. International Conf. of the IEEE Engineering in Medicine and Biology (EMBC'04)*, 2004, pp. 172-175.

[10] K. Nandan Babu and S. Saxena, "Parallel algorithms for the longest common subsequence problem", *Proc. 4th Intl. Conf. on High-Performance Computing*, Dec. 1997, pp. 120-125.

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

[12] J. Branke and M. Middendorf, "Searching for shortest common supersequences", *Proc. Second Nordic Workshop on Genetic Algorithms and Their Applications*, 1996, pp. 105-113.

[13] J. Branke, M. Middendorf and F. Schneider, "Improved heuristics and a genetic algorithm for fi nding short supersequences", *OR Spektrum* 20(1) (1998), pp. 39-46.

[14] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-Case Placers for Standard-Cell Layout," *Proc. ACM 1999 International Symposium on Physical Design (ISPD'99)*, pp. 90-96.

[15] A. Caldwell and A. Kahng and I. Markov, "Can Recursive bisection Produce Routable Designs?", *DAC*, 2000, pp.477-482.

[16] C. C. Chang, J. Cong and M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms", *Proc. Asia South-Pacific Design Automation Conference*, Jan. 2003.

[17] C.J. Colbourn, A.C.H. Lingi and M. Tompa, "Construction of optimal quality control for oligo arrays", *Bioinformatics* 18(4) (2002), pp. 529–535.

[18] J. Cong, M. Romesis and M. Xie, “Optimality, Scalability and Stability Study of Partitioning and Placement Algorithms”, *Proc. ISPD*, 2003, pp. 88-94.

[19] 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.

[20] V. Dancik, “Common subsequences and supersequences and their expected length”, *Combinatorics, Probability and Computing* 7(4) (1998), pp. 365-373.

[21] 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.

[22] J. J. Engel et al. ”Design methodology for IBM ASIC products”, *IBM Journal for Research and Development* 40(4) (1996), pp. 387.

[23] W. Feldman and P.A. Pevzner, “Gray code masks for sequencing by hybridization”, *Genomics*, 23 (1994), pp. 233–235.

[24] C. M. Fiduccia and R. M. Mattheyses, “A Linear-Time Heuristic for Improving Network Partitions”, *Proc. Design Automation Conference (DAC 1982)*, pp. 175–181.

[25] 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.

[26] D. E. Foulser, M. Li and Q. Yang, “Theory and algorithms for plan merging”, *Artificial Intelligence* 57(2-3) (1992), pp. 143-181.

[27] C. B. Fraser and R. W. Irving, “Approximation algorithms for the shortest common supersequence”, *Nordic J. Computing* 2 (1995), pp. 303-325.

[28] D.H. Geschwind and J.P. Gregg (Eds.), *Microarrays for the neurosciences: an essential guide*, MIT Press, Cambridge, MA, 2002.

[29] 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.

[30] L. W. Hagen, D. J. Huang and A. B. Kahng, “Quantifi ed Suboptimality of VLSI Layout Heuristics”, *Proc. ACM/IEEE Design Automation Conf.*, 1995, pp. 216–221.

[31] S. A. Heath and F. P. Preparata, “Enhanced Sequence Reconstruction With DNA Microarray Application”, *Proc. 2001 Annual International Conf. on Computing and Combinatorics (COCOON’01)*, pp. 64-74.

[32] D. J. Huang and A. B. Kahng, "Partitioning-Based Standard-Cell Global Placement with an Exact Objective", in *Proc. ACM/IEEE Intl. Symp. on Physical Design*, Napa, April 1997, pp. 18-25.

[33] 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.

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

[35] T. Jiang and M. Li, "On the approximation of shortest common supersequences and longest common subsequences", *SIAM J. on Discrete Mathematics* 24(5) (1995), pp. 1122-1139.

[36] L. Kaderali and A. Schliep, "Selecting signature oligonucleotides to identify organisms using DNA arrays", Bioiformatics 18:1340-1349, 2002.

[37] A.B. Kahng, I.I. Măndoiu, P.A. Pevzner, S. Reda, and A. Zelikovsky, "Border Length Minimization in DNA Array Design", *Proc. 2nd International Workshop on Algorithms in Bioinformatics (WABI 2002)*, R. Guigó and D. Gusfi eld (Eds.), Springer-Verlag Lecture Notes in Computer Science Series 2452, pp. 435-448.

[38] A.B. Kahng, I.I. Măndoiu, P.A. Pevzner, S. Reda, and A. Zelikovsky, "Engineering a Scalable Placement Heuristic for DNA Probe Arrays", *Proc. 7th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2003)*, W. miller, M. Vingron, S. Istrail, P. Pevzner and M. Waterman (Eds.), 2003, pp. 148–156.

[39] A.B. Kahng, I.I. Măndoiu, S. Reda, X. Xu, and A. Zelikovsky. Design flow enhancements for DNA arrays. In *Proc. IEEE International Conference on Computer Design (ICCD)*, pages 116–123, 2003.

[40] A.B. Kahng, I.I. Măndoiu, S. Reda, X. Xu, and A. Zelikovsky. Evaluation of placement techniques for DNA probe array layout. In *Proc. IEEE-ACM International Conference on Computer-Aided Design (ICCAD)*, pages 262–269, 2003.

[41] A.B. Kahng, I.I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky. Scalable heuristics for design of DNA probe arrays. *Journal of Computational Biology*, 11(2–3):429–447, 2004.

[42] 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 Research* vol. 30 (2002), e106.

[43] T. Kozawa et al., "Automatic Placement Algorithms for High Packging Density VLSI", *Proc. 20th Design Automation Conference (DAC 1983)*, pp. 175–181.

[44] F. Li and G.D. Stormo, “Selection of optimal DNA oligos for gene expression arrays,” Bioinformatics 17(11):1067-1076, 2001.

[45] R.J. Lipshutz, S.P. Fodor, T.R. Gingeras, D.J. Lockhart, “High density synthetic oligonucleotide arrays,” *Nature Genetics* 21 (1999), pp. 20–24.

[46] B. T. Preas and M. J. Lorenzetti (Eds.), *Physical Design Automation of VLSI Systems*, Benjamin-Cummings, 1988.

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

[48] S. Rahmann, “The Shortest Common Supersequence Problem in a Microarray Production Setting,” *Bioinformatics* 19 Suppl. 2 (2003), pp. 156-161.

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

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

[51] N.A. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Norwell, MA, 199

[52] L. Steinberg, “The backboard wiring problem: a placement algorithm”, *SIAM Review* 3 (1961), pp. 37–50.

[53] 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 Research* 30 (2002), e107.

[54] M. Wang, X. Yang and M. Sarrafzadeh, “DRAGON2000: Standard-cell Placement Tool For Large Industry Circuits”, *Proc. International Conference on Computer-Aided Design (ICCAD 2001)*, pp. 260–263.

[55] J.A. Warrington, R. Todd, and D. Wong (Eds.). *Microarrays and cancer research* BioTechniques Press/Eaton Pub., Westboro, MA, 2002.