Evaluation of Placement Techniques for DNA Probe Array Layout[*]

Andrew B. Kahng   Ion Măndoiu[†]   Sherief Reda   Xu Xu   Alex Z. Zelikovsky[‡]

CSE Department, University of California at San Diego
[†]CSE Department, University of Connecticut
[‡] CS Department, Georgia State University
E-mail: {abk,sreda,xuxu}@cs.ucsd.edu, ion@engr.uconn.edu, alexz@cs.gsu.edu

Abstract

DNA probe arrays 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, *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 a half million probes and are expected to exceed one hundred million probes in the next generation. This paper is one of the first attempts to apply VLSI CAD methods to the problem of probe placement in DNA chips, where the main objective is to minimize total *border cost* (i.e., the number of nucleotide mismatches between adjacent sites).

We make the following contributions. First, we propose several partitioning-based algorithms for DNA probe placement that improve solution quality by over 4% compared to best previously known methods. Second, we give a simple in-place probe reembedding algorithm with solution quality better than previous "chessboard" and batched greedy algorithms. Third, we experimentally evaluate scalability and suboptimality of existing and newly proposed probe placement algorithms. Interestingly, we find that DNA placement algorithms appear to have better suboptimality properties than those recently reported for VLSI placement algorithms [7, 8].

1 Introduction

DNA probe arrays – DNA arrays or DNA chips for short – have recently emerged as one of the core genomic technologies. They provide cost-effective means 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 [23] for a survey). Existing applications already cover many diverse fields ranging from healthcare to environmental sciences and law enforcement, and the number of applications is growing at an exponential rate [14, 27]. The rapid adoption of DNA arrays is due to a unique combination of robust manufacturing technology that leverages semiconductor wafer processes, massive parallel measurement capabilities, and highly accurate and reproducible results.

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, 13]. Similar to Very Large Scale Integration (VLSI) circuit manufacturing, multiple copies of a DNA array are simultaneously synthesized on a *wafer*, which is typically made out of quartz. When synthesis is complete, the wafers are diced and arrays are packaged individually. Depending on the number of distinct probes per array, a single $5'' \times 5''$ wafer can yield between 49 and 400 arrays.

The VLSIPS manufacturing process can be briefly described as follows. To initiate synthesis, linker molecules including a photolabile 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 photolabile protection group to prevent any 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.

Current commercial DNA arrays integrate hundreds of thousands of different probes on a surface only slightly larger than 1cm^2. Next-generation designs, enabled by the rapid scaling of the DNA array manufacturing processes into the sub-micron domain, are expected to integrate up to hundreds of millions of different probes [2, 23]. As the number and size of DNA array designs is expected to ramp up in coming years, there is an urgent need for high-quality software tools to assist in the design and manufacturing process. Existing design tools, dominated by ad-hoc heuristics with unknown suboptimality properties, are not well suited to handle the next generation of high-density arrays.

1.1 The Border Minimization Problem

Let $M_1, M_2, \ldots, M_K$ denote the sequence of masks used in the synthesis of an array, and let $s_i \in \{A, C, T, G\}$ be the nucleotide synthesized after exposing mask M_i. Every probe in the array must be a subsequence of the *nucleotide deposition sequence* $S = s_1 s_2 \ldots s_K$. In case a probe corresponds to multiple subsequences of S, one such subsequence, or "embedding" of the probe into S, must be chosen

[*]Work partially supported by Cadence Design Systems, Inc., the MARCO Gigascale Silicon Research Center, and NIH Award 1 P20 GM065762-01A1. The work of I.M. was performed while he was with the ECE Department at Univeristy of California, San Diego.

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.

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.[1] There are several types of synthesis errors that take place during array manufacturing. First, a probe may not lose 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 on the chip 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 the use of smaller array sites and therefore the integration of a larger number of probes per array.

Formally, the border minimization problem is equivalent to finding a *three-dimensional placement* of the probes [18]: two dimensions represent the site array, and the third dimension represents the nucleotide deposition sequence S (see Figure 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 three-dimensional placement representation. Border length of a given mask is computed as the number of *conflicts*, i.e., pairs of adjacent exposed and masked sites in the mask. Given two adjacent embedded probes p and p', the *conflict distance* $d(p,p')$ is the number of conflicts between the corresponding columns. The 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.

A special case is that of a *synchronous* synthesis regime, in which the nucleotide deposition sequence S is periodic, and the k^{th} period $(ACGT)$ of S is used to synthesize a single (the k^{th}) additional nucleotide in each probe. Since in this case the embedding of a probe is predefined, the problem reduces to finding a *two-dimensional 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.

[1]DNA chips are used for performing *hybridization experiments*. In a hybridization experiment, a DNA chip is exposed to a solution containing fluorescently labeled fragments of DNA. These fragments are strings over the alphabet $\{A,C,G,T\}$ that *hybridize*, i.e., bind, to complementary probe strings according to the rule that A binds to T (and vice-versa), and C binds to G (and vice-versa). Copies of a given string in the solution will hybridize to copies of the complementary probe (if present) on the DNA chip; this can be discerned by measuring fluorescence intensity over the DNA probe array after the experiment has been completed.

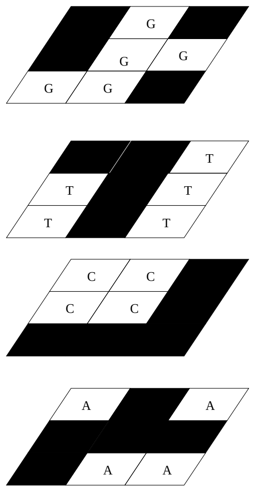


Figure 1: 3-dimensional probe placement with 4 masks and $S = ACTG$. Total border length is 24 (7 on the A mask, 4 on the C mask, 6 on the T mask, and 7 on the G mask).

1.2 Previous Work

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 [11], who proposed an optimal solution based on 2-dimensional Gray codes. Hannenhalli et al. [16] 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 two-dimensional array of sites, using a technique similar to one previously used in VLSI design [20]. For the same synchronous context, [18] suggested an *epitaxial*, or "seeded crystal growth", placement heuristic similar to heuristics explored in the VLSI circuit placement literature by [24, 25].

The general border minimization problem, which allows arbitrary *asynchronous* probe embeddings, was introduced by Kahng et al. in [18]. 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 re-placing them. Very recently, [19] proposed methods with near-linear runtime combining simple ordering-based heuristics for *initial placement*, including lexicographic sorting followed by threading, with heuristics for *placement improvement*, optimal reassignment of an "independent" set of probes [26] chosen from a sliding window [10], and a row-based implementation of the epitaxial algorithm in [18] that speeds-up the computation by considering only a limited number of candidates when filling each array site.[2]

1.3 Contributions of this Work

In this paper, we make several contributions. First, we propose the first *partitioning-based* algorithms for DNA probe placement (Section 2). Our experimental results show that the partitioning-based approach has better solution quality and/or runtime compared

[2]The work of [19] also extends probe placement algorithms 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.

```
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 $\{C_0, C_1, C_2, C_3\}$
```

Figure 2: The *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 $p_{i,j} = q$
    $Q = Q \setminus q$
```

Figure 3: The *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 $|Q| = 20000$ through a *borrowing* heuristic.

to previous methods (Section 3.1). Second, we give a simple in-place probe re-embedding algorithm that gives better solution quality than the "chessboard" and batched greedy algorithms proposed in [18] (Section 3.2). Third, we experimentally study the scalability and suboptimality of existing and newly-proposed DNA probe placement algorithms (Section 3.3). We observe that border cost normalized by the number of pairs of adjacent array sites is decreasing with array size for all studied algorithms (whereas it remains constant for a random placement). This is an encouraging observation for future scaling of the DNA array technology. Furthermore, we augment lower bound-based suboptimality studies of [18] by introducing two new techniques for quantifying the suboptimality of a placement heuristic: (a) comparison to known optimum solutions for specially structured instances, and (b) computation of the "scaling suboptimality" coefficient [15] that measures the growth of solution cost on scaled instances. The results of our study indicate that, in general, DNA placement algorithms have better suboptimality properties than those recently reported for VLSI placement algorithms in [7, 8].

2 Partitioning-Based Probe Placement

Recursive partitioning has been the basis of numerous successful VLSI placement algorithms [5, 6, 22] 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 hyperedge cut between these sets. The min-cut goal is typically achieved through the use of the Fiduccia-Mattheyses procedure [12], often in a multilevel framework [6]. Unfortunately, direct transfer of the recursive min-cut placement paradigm from VLSI to VLSIPS is blocked by the fact

```
Input: Chip size $S \times S$; set $P$ of DNA probes
Output: Probe placement which heuristically minimizes total conflicts
---
Let $l = 0$ and let $L =$ maximum recursion depth
Let $R^l_{1,1} = P$
For $l = 0$ to $L - 1$
  For $i = 1$ to $2^l$
    For $j = 1$ to $2^l$
      Let the set of (next-level) subpartitions be $R_{next} =$
          $\{R^{l+1}_{2i-1,2j-1} = \emptyset, R^{l+1}_{2i-1,2j} = \emptyset, R^{l+1}_{2i,2j-1} = \emptyset, R^{l+1}_{2i,2j} = \emptyset\}$
      SelectCentroid($R_{next}$)
      For all probes $p \in R^l_{i,j}$
        Insert $p$ into the yet-unfilled partition of $R_{next}$ 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 4: Partitioning-based DNA probe placement heuristic.

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 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.[3] 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 maximum total distance among themselves, then use these as the centroids. This procedure, described in Figure 2, is reminiscent of the *k*-center approach to clustering studied by Alpert et al. [4], and of methods used in large-scale document classification [9].

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 of [19], which for completeness of exposition is replicated in Figure 3.

The complete partitioning-based placement algorithm for DNA arrays is given in Figure 4. At a high level, our method resembles global-detailed approaches in the VLSI CAD literature [17, 21]. The algorithm recursively quadrisects every partition at a given level, assigning the probes so as to minimize distance to the centroids of subpartitions.[4] In the innermost of the three nested *for* loops of Figure 4, we apply a multi-start heuristic, trying r different random probes as seed C_0 and using the result that minimizes total distance to the centroids. Once the maximum level L of the recur-

[3]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.

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

sive partitioning is attained, detailed placement is executed via the row-epitaxial algorithm. Additional details and commentary are as follows.

- Our construction of partitions has an obvious dependence on the order in which probes p are considered. Our work has not yet examined the impact of the probe ordering degree of freedom on centroid-based partitioning.
- 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 "round" clusters, but can also correct for unfortunate choices of the initial four centroids.[5]
- The straightforward implementation of *Reptx()*-based detailed placement within a given partition will treat the last locations "unfairly", e.g., for the last location considered, be only one candidate probe will remain. To balance the options for every position, our implementation permits "borrowing" probes from the next region in the *Reptx()* procedure. For every position, we select the best probe from at most m probes, where m is a pre-determined constant, in the current region *and the next region*. (Except as noted, we set m to 20000 for all of our experiments.) Our *Reptx()* implementation is also "border-aware", that is, it takes into account Hamming distances to the placed probes in adjacent regions.

2.1 Time Complexity

Let the number of probes in a chip be n. The procedure *Select-Centroid()* for a partitioned region at recursion depth l takes $O(\frac{n}{4^l})$ steps, and grouping all the probes into four partitions also takes $O(\frac{n}{4^l})$ steps. Therefore, the runtime for every recursion depth l is $O(\frac{n}{4^l}4^l) = O(n)$. Since there are L recursion depths, the overall runtime for partitioning is $O(Ln)$. For the *Reptx()* procedure, *at most* $m = 20000$ comparisons are executed for every position. Therefore, the total runtime is $O(n(L+m))$. Since $L \leq \log_4 n$, the runtime is $O(n(\log_4 n + m))$.

3 Comparison of Probe Placement Heuristics

3.1 Partitioning-Based vs. Other Methods

We compare four two-dimensional placement algorithms.

1. **TSP+Threading [16]:** This algorithm computes a TSP tour in the complete graph with the probes as vertices and edge costs given by Hamming distances. The tour is then *threaded* into the two-dimensional array of sites using the 1-threading method described in [16].
2. **Row-Epitaxial [19]:** An implementation of the epitaxial algorithm in [18], 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 20000 in our experiments.
3. **Sliding-Window Matching (SWM) [19]:** 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 re-placing them using a minimum-weight perfect matching algorithm (cf. "row-ironing" [6]).
4. **Recursive partitioning based placement:** As described in Section 2 above.

Table 1 compares the results produced by the first three (previously known) algorithms on random instances with chip sizes between 100 and 500 and probe length equal to 25. We find that Row-Epitaxial is the algorithm with highest solution quality, while SWM is the fastest, offering competitive solution quality with much less runtime. Besides total border cost, we also report the border cost normalized by the number of pairs of adjacent array sites, i.e., we also give the *average* number of conflicts per pair of adjacent sites. This number decreases with increasing chip size, which can be attributed to greater freedom of choice available when placing a higher number of probes.

Table 2 presents results for our new recursive partitioning method with different maximum recursion depths $L = 1, 2, 3$. Comparing to the results produced by Row-Epitaxial, the best previously known technique (Table 1), we find that recursive partitioning based placement achieves on the average similar or better results with improved runtime.

3.2 A Note on In-Place Re-Embedding

While VLSI physical design automation may be viewed as primarily a "Place & Route" process, VLSIPS (DNA array) physical design automation is rather a "Place & Embed" process. Recall that *embedding* exploits the possibility of *asynchronous* embedding of probes within the mask sequence to further reduce the total number of conflicts. Here, we note our development of a new in-place re-embedding algorithm – i.e., a method which, given a two-dimensional probe placement, improves the embedding of the probes without re-placing them – and compare the new method against two previous in-place re-embedding algorithms.

Each of the following 3 algorithms uses the dynamic programming algorithm in [18] for optimal re-embedding of a probe with respect to the embeddings of its neighbors:

1. **Batched greedy [18]:** 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 non-adjacent probes is selected for re-embeddeding by traversing the probes in non-increasing order of gain.
2. **Chessboard [18]:** In this method, the 2-dimensional 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 an 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).
3. **Sequential:** Our new algorithm performs optimal re-embedding of probes in a sequential row-by-row fashion. We believe that a main shortcoming of Batched Greedy and

[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 $d(p, C_i) = \frac{1}{N_i} \sum_t \sum_{s \neq p[t]} N_{s,t}$.

Chip	Lower Bound		TSP+1Thr				Row-Epitaxial				SWM			
Size	Cost	Norm.	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
200	1512014	19.0	2140903	26.9	41.6	1901	1913796	24.0	26.6	1151	2360540	29.7	56.1	8
300	3233861	18.0	4667882	26.0	44.3	12028	4184018	24.0	29.4	3671	5192839	28.9	60.6	19
500	8459958	17.0	12702474	25.5	50.1	109648	11182346	22.4	32.2	10630	13748334	27.6	62.5	50

Table 1: Total border cost, normalized border cost, gap from (synchronous placement) lower-bound [18], and CPU seconds (averages over 10 random instances) for the TSP heuristic of [16] (TSP+1Thr), and the row-epitaxial (Row-Epitaxial) and sliding-window matching (SWM) heuristics of [19]. We use an upper bound of 20000 on the number of candidate probes in Row-Epitaxial, and 6×6 windows with overlap 3 for SWM.

Chip	Lower Bound		RPART $L = 1$				RPART $L = 2$				RPART $L = 3$			
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	23.4	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 2: Total border cost, normalized border cost, gap from (synchronous placement) lower-bound [18], and CPU seconds (averages over 10 random instances) for the recursive partitioning algorithm with recursion depth varying between 1 and 3.

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

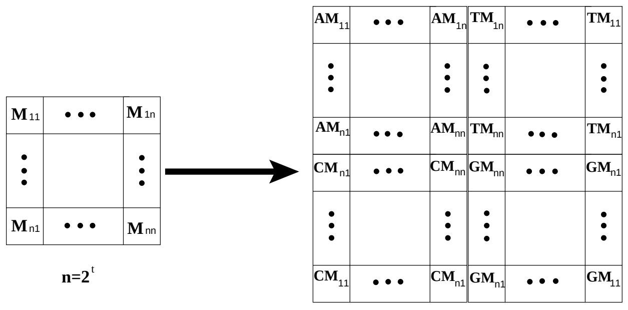
Table 3: Total border cost, normalized border cost, gap from *asynchronous post-placement* lower-bound [18], and CPU seconds (averages over 10 random instances) for the batched greedy, chessboard, and sequential in-place re-embedding algorithms.

Chip	Lower Bound		TSP+1Thr				Row-Epitaxial				SWM			
Size	Cost	Norm.	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
200	798708	10.0	1723352	21.7	115.8	1901	1608382	20.2	101.4	1368	1721633	21.6	115.6	380
300	—	—	3801765	21.2	—	12028	3529745	20.3	—	3861	3801479	21.2	—	861
500	—	—	10426237	20.9	—	109648	9463941	19.0	—	12044	10161979	20.4	—	2239

Table 4: Total border cost, normalized border cost, gap from *asynchronous pre-placement* lower-bound [18], and CPU seconds (averages over 10 random instances) for the TSP heuristic of [16] (TSP+1Thr), and the row-epitaxial and sliding-window matching heuristics of [19] (Row-Epitaxial and SWM), each followed by sequential in-place re-embedding. We use an upper bound of 20k on the number of candidate probes in Row-Epitaxial and 6×6 windows with overlap 3 for SWM.

Chip	Lower Bound		RPART $L = 1$				RPART $L = 2$				RPART $L = 3$			
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

Table 5: Total border cost, normalized border cost, gap from *asynchronous* lower-bound [18], and CPU seconds (averages over 10 random instances) for the recursive partitioning algorithm followed by sequential in-place re-embedding.



M_{11} ••• M_{1n}
M_{n1} ••• M_{nn}
$n=2^t$
AM_{11} ••• AM_{1n} TM_{1n} ••• TM_{11}
AM_{n1} ••• AM_{nn} TM_{nn} ••• TM_{n1}
CM_{n1} ••• CM_{nn} GM_{nn} ••• GM_{n1}
CM_{11} ••• CM_{n1} GM_{n1} ••• GM_{11}

Figure 5: 2-dimensional Gray code placement.

Chessboard is that these methods always re-embed an *independent* set of sites on the DNA chip. Our intuition is that dropping this requirement permits faster propagation of the effects of any new embedding, and hence convergence to a better local optimum.

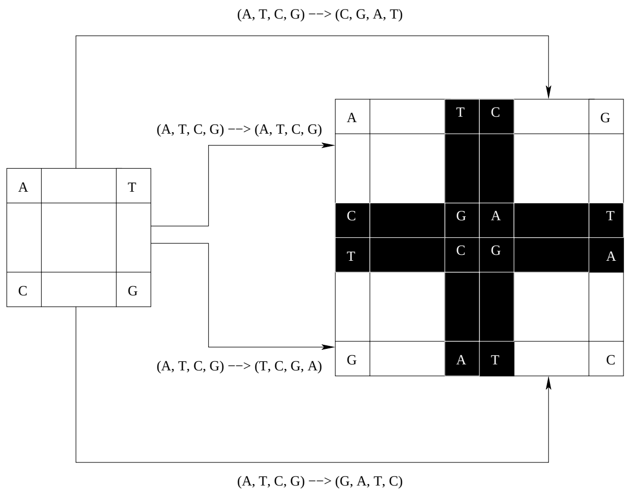
Table 3 compares the three algorithms on random instances with chip sizes between 100 and 500 and probe length 25. The initial two-dimensional placements were obtained using TSP+1-threading. All algorithms are stopped when the improvement cost achieved in one iteration over the whole chip drops below 0.1% of the total cost. The results show that re-embedding of the probes in a sequential row-by-row order leads to a reduction in the border cost by 0.8% compared to the chessboard algorithm.

3.3 Comparison of Placement and Embedding Flows

Another series of experiments executes the complete placement and embedding flow. We compare the four two-dimensional placement algorithms discussed in Section 3.1, when each is followed by sequential in-place re-embedding. Results are given in Tables 4-5. Table 4 compares existing methods: TSP+1Thr, row-epitaxial (REPTX) with 20000 lookahead probes and sliding window matching (SWM) with 6 x 6 windows and an overlap of 3. TSP+1Thr is dominated by both REPTX and SWM in both conflict cost and running time. REPTX produces less conflicts than SWM but SWM is considerably faster. This tradeoff between speed and quality is, of course, reminiscent of the VLSI CAD experience with heuristics for difficult optimizations. Table 5 gives the number of conflicts for the proposed recursive quadrisection partitioning technique. L indicates the recursion depth for which the given results are obtained. Comparing these results to those in Table 4, we see that DNA chip placement using recursive-partitioning outperforms the best previous flow (row-epitaxial + sequential re-embedding) by an average of 4.0%.

3.4 Quantified sub-optimality of placement and embedding algorithms

As noted in the introduction, next-generation DNA probe arrays will contain up to one hundred 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. [15], and recently extended by Chang et al. [7] and Cong et al. [8]. In this framework, there are two basic types of instance scaling that we can apply.



(A, T, C, G) --> (C, G, A, T)
(A, T, C, G) --> (A, T, C, G)
(A, T, C, G) --> (T, C, G, A)
(A, T, C, G) --> (G, A, T, C)
A
T
C
G
A
T
C
G
C
G
A
T
T
C
G
A
G
A
T
C

Figure 6: Scaling construction used in the suboptimality experiment.

- **Instances with known optimum solution.** For hypergraph placement, instances with known minimum-wirelength solutions may be constructed by "overlaying" signal nets within an already placed cell layout, such that each signal net has provably minimum length. This technique, proposed by Boese and explored by Chang et al. [7], induces a netlist topology with prescribed degree sequence over the (placed) cells; this corresponds to a "placement example with known optimal wirelength" (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-dimensional Gray codes [11]. Our construction generates 4^k probes which are placeable such that every probe has border cost of 1 to each of its neighboring probes. This construction is illustrated in Figure 5.

- **Instances with known suboptimal solutions.** Because constructed instances with known optimum solutions may not be representative of "real" instances, we also apply a technique [15] that allows real instances to be scaled, such that they offer insights into scaling of heuristic suboptimality. A technique of Hagen et al. is applied as follows. Beginning with an instance I consisting of a ("real") DNA chip probe set, we induce 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 6. If a heuristic H places I' with cost $c_H(I') \geq 4 \cdot c_H(I)$, then we may infer 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 at least on this class of instances, the heuristic's solution quality would be said to scale well.

Table 6 shows results from executing the various placement heuristics on PEKO-style test cases, with instance sizes ranging from 16 x 16 through 512 x 512 (recall that our Gray code construction yields instances with 4^k probes). We see from these results that sliding-window matching approaches closest to the optimal values, with a suboptimality gap of around 30%. Overall, DNA array placement algorithms appear to be performing better than their VLSI counterparts [7] 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.* Our results, though, do illustrate this point: algorithms that perform best for arbitrary benchmarks are not necessarily the best performers for specially constructed benchmarks.

Table 7 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 x 100 to 500 x 500, and the scaled instances thus ranging in size from 200 x 200 to 1000 x 1000. This table shows that in general, placement algorithms for DNA arrays offer excellent suboptimality scaling. 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 total conflict well under the constructive upper bound.

4 Conclusions

DNA probe arrays (or DNA chips) are an important technology that provides a cost-effective vehicle for genomic analyses and high-value demand for silicon processing technology. In this work, we have studied the transfer of VLSI CAD physical design automation techniques to DNA chips, focusing on minimizing the total border length between adjacent sites during probe placement and embedding. Our main contributions are:

- Drawing a fertile analogy between DNA array and VLSI design automation, with DNA Place & Embed corresponding to VLSI Place & Route.
- Proposing 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 probe embedding processing technique for asynchronous re-embedding of placed probes within the mask sequence. Experimental results show that combining the new algorithms results in average improvement of 4.0% over best previous flows.
- Studying and quantifying the performance of existing and newly proposed DNA Place & Embed algorithms in experiments on benchmarks with known optimal cost as well as *scaling suboptimality* experiments, in a manner similar to recent studies in the VLSI CAD field.

We conclude with some remarks that address observed similarities and contrasts between VLSI placement and VLSIPS placement. 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. In fact, our experiments show that the *gap* between the lower bound of [18] and the actual number of conflicts shrinks as chip size increases. Second, the lower bounds for DNA Place & Embed appear to be tighter than those available in the VLSI placement literature. Developing even tighter lower bounds is, of course, an important open problem.

Finally, taking a broader perspective, we can envision emerging DNA CAD flows that are similar to VLSI CAD counterparts. The main steps of a "back-end" VLSIPS flow might be as follows:

1. **Probe Selection:** Analogous to logic synthesis in VLSI design, this step is responsible for implementing the desired functionality of the DNA array.
2. **Deposition Sequence Design:** This optimization step is responsible for minimizing the number of synthesis steps, or equivalently, minimizing the number of photolithographic masks used in the manufacturing process.
3. **Design of Control and Test Structures:** DNA array test structures are the equivalent of built-in-self-test (BIST) structures in VLSI design, and aim at detecting manufacturing defects.
4. **Probe Placement and Embedding:** As we discussed in this work, this step is equivalent to the physical design step in VLSI CAD.

Enhancing this basic flow by adding flow-awareness to each optimization step and introducing feedback loops is an important direction for future research.

References

[1] http://www.affymetrix.com

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

[3] S.B. Akers, "On the use of the linear assignment algorithm in module placement", *Proc. 18th Design Automation Conference (DAC 1981)*, 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, "Recent directions in netlist partitioning: A survey", *Integration: The VLSI Jour.* 19 (1995), pp. 1-81.

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

[7] 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.

[8] J. Cong, M. Romesis and M. Xie, "Optimality, Scalability and Stability Study of Parititoning and Placement Algorithms", *Proc. ISPD*, 2003, pp. 88-94.

[9] 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.

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 6: Comparing the placement algorithms performance for cases with known optimal conflicts. SW matching is using a window size of 20 x 20 and a step of 10. Row-epitaxial uses $10000/chipsize$ 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

Table 7: 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 x 20 and a step of 10. Row epitaxial uses $10000/chipsize$ lookahead rows.

[10] 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.

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

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

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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. Gusfield (Eds.), Springer-Verlag Lecture Notes in Computer Science Series 2452, pp. 435–448.

[19] 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.

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

[21] M. Sarrafzadeh and M. Wang, "NRG: GLobal and Detailed Placement", *Proc. International Conference on Computer-Aided Design (ICCAD 1997)*, pp. 532–537.

[22] 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.

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

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

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

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

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