UNIVERSITY OF CALIFORNIA, SAN DIEGO

New Approaches for Placement and Benchmarking of CMOS and Gene Chips

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science (Computer Engineering)

by

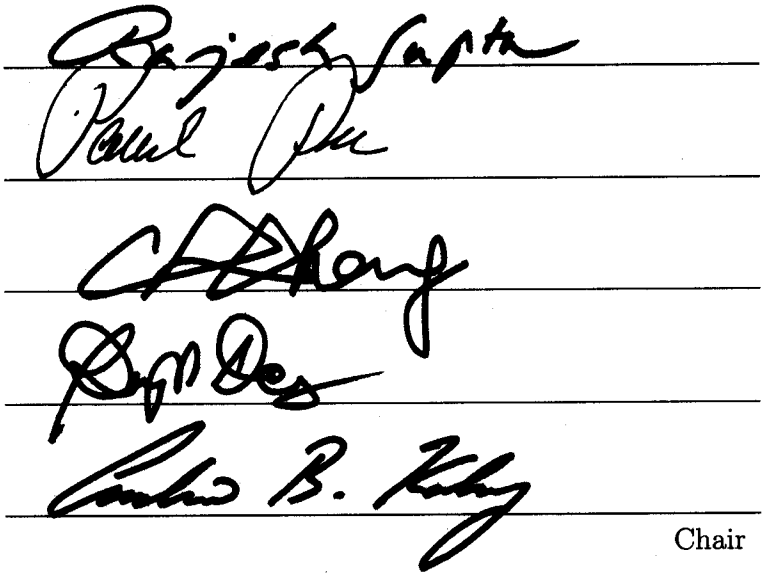
Sherief Mohamed Reda

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Sujit Dey
Professor Rajesh Gupta
Professor Pavel Pevzner

2006

The dissertation of Sherief Mohamed Reda is approved, and it is acceptable in quality and form for publication on microfilm:



Chair

University of California, San Diego

2006

To people who seek to learn and understand. May this thesis enlighten your search.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to thank my family (my father Mohamed Reda, my mother Makarem, and my sister Dalia) for their unconditional support, love, and encouragement throughout the years. Without doubt, their continuous nurturing of my confidence has been the most important contribution to my life. I am certainly indebted to them; they are the ones who felt the impact of my prolonged absence.

I certainly feel privileged and grateful to work under Professor Andrew Kahng's guidance, and I thank him for all his invaluable support and advice. Without doubt, Professor Kahng has introduced me to new patterns of creative, effective thinking, and it is always inspiring to observe Professor Kahng's energy. I also thank him for the freedom he gave me in pursing different research ideas. Indeed, Professor Kahng's laboratory is a thriving research environment, and I have no doubt that it would have been impossible to pursue the scope and diversity of work I did with him anywhere else.

I would like to thank Professor Pavel Pevzner for all his help and advice. Professor Pevzner has been an inspiration, and together with Professor Kahng suggested working on Gene chip problems – a significant part of this thesis. Professor Pevzner has also helped me with my career choices, and I am looking forward to further successful projects with him in the future.

I would also like to thank my thesis committee members, Professor C. K. Cheng, Professor Sujit Dey, and Professor Rajesh Gupta, for taking time out of their busy schedules to review and evaluate my research work. I am quite grateful for their valuable feedback.

My early fruitful collaboration with Professor Ion Măndoiu was a key to broadening my thinking, and I would like to thank him for his sincere advice and support. I also thank Professor Igor Markov for his insightful comments about my research on placement. His comments were quite important in avoiding pitfalls. Professor Markov has also helped me on numerous occasions, in a variety of contexts. I would also like to thank Prof. Alex Zelikovsky for our earlier collaboration. I am also quite grateful to Prof. Alex Orailoglu who helped with the admission

process to UCSD, and for his mentorship during my first year in the Ph.D. program. My Masters advisor, Professor Ashraf Salem from Ain Shams University, has been continuously supporting and encouraging me throughout the years. I am quite indebted to him.

Our department coordinator, Julie Conner, certainly deserves special thanks for her constant, energetic advice and support. Julie has always been helpful and quite instrumental in keeping my Ph.D. on track. I also thank Virginia McIlwain, our lab's administrator, for all her help, encouragement, and care in the past three years. Virginia handled our group administrative tasks superbly.

I certainly feel lucky to work with a group of bright fellows – Puneet Sharma, Puneet Gupta, Swamy Muddu, Kambiz Samadi, Xu Xu, Bao Liu, Qinke Wang, and Chul-Hong Park – in Professor Kahng's laboratory. We had wonderful projects, and we spent great times together. I wish them the best in their future. I would also like to thank my friends: Bo-Bakr Drissi, Mohamed Jalloh, Wael El-Dahshan, and Khaled Essa, for all the fun and the great time we spent together. They have been quite supportive.

Last, but not least, I would like to thank Kimberly Harada for all her support, encouragement, and love. Her well-balanced character has been a key in stabilizing my life in the past year and a half, and her delicious cooking has been nurturing my body as well. I am looking forward to further wonderful times with her.

The material in this thesis is based on the following publications. Authors' names are listed in alphabetical order in all publications.

- Chapter II is based on the following publication: A. B. Kahng and S. Reda, "Zero-Change Netlist Transformations: A New Technique for Placement Benchmarking," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

- Chapter III is based on the following publication: A. B. Kahng and S. Reda, "Wirelength Minimization for Min-Cut Placements via Placement

Feedback," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

- Chapter IV is based on the following publication: A. B. Kahng, I. Mandoiu, S. Reda, A. Zelikovsky and X. Xu, "Computer-Aided Optimization of DNA Array Design and Manufacturing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25(2), 2006, pp. 305 – 320.

- Chapter V is based on the following publication: A. B. Kahng, I. Mandoiu, P. Pevzner, S. Reda and A. Zelikovsky, "Scalable Heuristics for Design of DNA Probe Arrays," *Journal of Computational Biology*, Vol. 11(2-3), 2004, pp. 429 – 447.

The dissertation author was the primary researcher and author. My coauthors (Prof. Andrew B. Kahng, Prof. Pavel Pevzner, Prof. Ion Mandoiu, Prof. Alex Zelikovsky, and Mr. Xu Xu) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

1975	Born, Oslo, Norway
1998	B.Sc. Electrical and Computer Engineering, Ain Shams University, Cairo, Egypt
2000	M.Sc. Electrical and Computer Engineering, Ain Shams University, Cairo, Egypt
2004	C.Phil., Computer Science (Computer Engineering), University of California, San Diego
2006	Ph.D., Computer Science (Computer Engineering), University of California, San Diego

SELECTED PUBLICATIONS

All papers coauthored with my advisor Prof. Andrew B. Kahng have authors listed in alphabetical order.

- A. Kahng and S. Reda, "Zero-Change Netlist Transformations: A New Technique for Placement Benchmarking," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
- A. Kahng and S. Reda, "Wirelength Minimization for Min-Cut Placements via Placement Feedback," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.
- C. Alpert, A. Kahng, G.-J. Nam, S. Reda and P. Villarubia, "A Fast Hierarchical Quadratic Placement Algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25(4), 2006, pp. 678 – 691.
- A. Kahng and S. Reda, "New and Improved BIST Diagnosis Techniques from Combinatorial Group Theory," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25(3), 2006, pp. 533 – 543.

- A. Kahng, I. Mandoiu, S. Reda, A. Zelikovsky and X. Xu, "Computer-Aided Optimization of DNA Array Design and Manufacturing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25(2), 2006, pp. 305 – 320.

- A. Kahng and S. Reda, "Match Twice and Stitch: A New TSP Tour Construction Heuristic," *Operations Research Letters*, Vol. 32(6), 2004, pp. 449 – 509. **"Hot" (most downloaded) article in Operations Research Letters, February 2005.**

- A. Kahng, I. Mandoiu, P. Pevzner, S. Reda and A. Zelikovsky, "Scalable Heuristics for Design of DNA Probe Arrays," *Journal of Computational Biology*, Vol. 11(2-3), 2004, pp. 429 – 447.

- A. Kahng and S. Reda, "A Tale of Two Nets: Studies in Wirelength Progression in Physical Design," *Proc. System-Level Interconnect Prediction Workshop,* 2006, pp. 17 – 24.

- S. Reda and A. Chowdhary, "Effective Linear Programming Based Placement Methods," *Proc. International Symposium on Physical Design,* 2006, pp. 186 – 191.

- A. Kahng and S. Reda, "Intrinsic Shortest Path Length: A New, Accurate A Priori Wirelength Estimator," *Proc. International Conference on Computer-Aided Design,* 2005, pp. 173 – 180.

- A. Kahng, S. Reda and Q. Wang, "Architecture and Details of a High Quality, Large-Scale Analytical Placer," *Proc. International Conference on Computer-Aided Design*, 2005, pp. 891 – 898. **Best paper nomination.**

- Y. Cheon, P.-H. Ho, A. Kahng, S. Reda and Q. Wang, "Power-Aware Placement," *Proc. Design Automation Conference*, 2005, pp. 795 – 800.

- A. Kahng and S. Reda, "Evaluation of Placer Suboptimality via Zero-Change Transformations," *Proc. International Symposium on Physical Design*, 2005,

pp. 208 – 215.

- C. Alpert, A. Kahng, G.-J. Nam, S. Reda and P. Villarubia, "A Semi-Persistent Clustering Technique for VLSI Circuit Placement," *Proc. International Symposium on Physical Design*, 2005, pp. 200 – 207.

- A. Kahng, S. Reda and Q. Wang, "APlace: A General Analytic Placement Framework," *Proc. International Symposium on Physical Design*, 2005, pp. 233 – 235. **Invited. Winner of the ISPD-2005 placement contest.**

- A. Kahng and S. Reda, "Reticle Floorplanning With Guaranteed Yield for Multi-Projects Wafer," *Proc. International Conference on Computer Design*, 2004, pp. 106 – 110.

- A. Kahng and S. Reda, "Placement Feedback: A Concept and Method for Better Min-Cut Placements," *Proc. Design Automation Conference*, 2004, pp. 357 – 362.

- A. Kahng, I. Markov and S. Reda, "On Legalization of Row-Based Placements," *Proc. Great Lakes VLSI Symposium*, 2004, pp. 214 – 219.

- A. Kahng, I. Markov and S. Reda, "Boosting: A Min-Cut Placement with Improved Signal Delay," *Proc. Design Automation and Test in Europe*, 2004, pp. 1098 –1103.

- A. Kahng and S. Reda, "Combinatorial Group Testing Methods for the BIST Diagnosis Problem," *Proc. Asia South Pacific Design Automation Conference*, 2004, pp. 113 – 116.

- A. Kahng, I. Mandoiu, S. Reda, X. Xu and A. Zelikovsky, "Evaluation of Placement Techniques for DNA Probe Array Layout," *Proc. International Conference in Computer-Aided Design*, 2003, pp. 262 – 269.

- A. Kahng, I. Mandoiu, S. Reda, X. Xu and A. Zelikovsky, "Design Flow Enhancements for DNA Arrays," *Proc. International Conference on Computer Design*, 2003, pp. 116 – 123.

- A. Kahng, I. Mandoiu, P. Pevzner, S. Reda and A. Zelikovsky, "Engineering a Scalable Placement Heuristic for DNA Probe Arrays," *Proc. International Conference on Research in Computational Molecular Biology*, 2003, pp. 148 – 156.

- A. Kahng, I. Mandoiu, P. Pevzner, S. Reda and A. Zelikovsky, "Border Length Minimization in DNA Array Design," *Proc.* 2^{nd} *Workshop on Algorithms in Bioinformatics*, 2002, R. Guigo and D. Gusfield (eds.), Springer-Verlag Lecture Notes in Computer Science Series 2452, pp. 435–448.

ABSTRACT OF THE DISSERTATION

New Approaches for Placement and Benchmarking of CMOS and Gene Chips

by

Sherief Mohamed Reda
Doctor of Philosophy in Computer Science (Computer Engineering)
University of California, San Diego, 2006
Professor Andrew B. Kahng, Chair

The importance of component placement, for both Complementary Metal Oxide Semiconductor (CMOS) chip and Gene chip (or DNA array) technologies, cannot be underestimated. Placement determines (a) the physical performance of Very Large Scale Integration (VLSI) CMOS circuits, and (b) the level of integration and fidelity of Gene chips. Benchmarking is concerned with calculating the gap between the result of a suboptimal algorithm and the result of the (generally unachievable) optimal algorithm. Any calculated difference fuels continued attention to the given problem. In the framework of placement and benchmarking, this thesis is concerned with two central questions. (1) Are current placement algorithms optimal (or "near" optimal) with respect to realistic CMOS and Gene chip instances? And (2) If question (1) is answered negatively, how can the performance of placement algorithms be improved?

In the context of CMOS chips, this thesis proposes a new approach, *zero-change netlist transformations* (ZCNT), for placement benchmarking. ZCNT answers question (1) negatively, and proves, for the first time, that placers display significant suboptimal behavior (by up to 22%) for circuit instances that are realistic. This is in contrast to previous approaches which gave results on highly artificial circuit topologies or reported loose lower bounds. Then in answer to question (2), this thesis proposes a new placement technique, *placement feedback*, which improves top-down placement frameworks. Implementing the proposed technique in the open-source placer Capo yields placements with up to 15% improvement in wirelength.

In the context of Gene chips, this thesis proposes a number of benchmarking methods, including calculation of placement lower bounds and construction of chips with known optimal (and suboptimal) placements. The proposed benchmarking methods answer question (1) negatively by proving that current Gene chip placement algorithms are suboptimal by about 42%. Having answered question (1) negatively, the thesis answers question (2) by proposing a number of new placement approaches – constructive and iterative in nature – that improve upon current algorithmic approaches. The effectiveness of the proposed approaches is demonstrated by providing placements for academic chips and for an industrial Human Genome chip that are better than previous approaches by 18% and 6% respectively.

Chapter I

Introduction

Complementary Metal Oxide Semiconductor (CMOS) and Gene chips are the cornerstones of technologies in the areas of computational electronics and molecular biology. These chips have remarkable similarities and differences with respect to several key perspectives. From a fabrication perspective, CMOS chips and Gene chips are manufactured using similar optical lithographic techniques[1]. Thus, both suffer from the detrimental effects of light diffraction, and share the potential benefit of scaling by reduction of feature sizes. From an application perspective, the two chips are completely different – CMOS chips perform computations, while Gene chips perform a parallel search to probe or interrogate a sample of RNA fragments for the presence of given content. From a structural perspective, both chips are composed of components – logic gates, I/O pads, and memory blocks in case of CMOS chips, and DNA strands or probes in case of Gene chips – that have to be *placed* on a two-dimensional surface to optimize a certain objective. One notable difference between CMOS and Gene chips is that CMOS chips have wires that connect various structural components to distribute the results of computations.

The placement of components of CMOS and Gene chips is the focus of this thesis. We will motivate and define placement in this chapter. Afterward, the outline of the thesis is simple. For each of these technologies (CMOS and

[1]At least for Gene chips with the leading manufacturer Affymetrix.

Gene chips), I will first prove – using new, proposed benchmarking techniques – that existing placement approaches are suboptimal and have substantial room for improvement. Afterward, I will then propose a number of improved placement approaches that reduce the amount of suboptimality and lead to better chips. The rest of this chapter introduces and defines the CMOS and Gene chip placement contexts.

I.A Introduction to CMOS Chip Placement

Placement is an essential step in the physical design flow of Very Large Scale Integration (VLSI) CMOS chips since it assigns exact locations to various circuit components within the chip's core area. An inferior placement assignment will not only affect the chip's performance but might also render it non-manufacturable, by requiring excessive wirelength beyond available routing resources. Consequently, a placer must perform the assignment while optimizing a number of objectives to ensure that a circuit meets its performance demands. Typical placement objectives include total wirelength, timing, routing congestion, and power.

Figure III.3 shows the position of placement within a typical CMOS chip design flow. A placer takes a given synthesized circuit netlist together with a technology library and produces a valid placement layout. The layout is optimized according to the aforementioned objectives and is then ready for the ensuing cell re-sizing and buffering – a step essential for timing and signal integrity constraint satisfaction. Clock tree synthesis and routing follow, completing the physical design process. In many cases, parts of, or the entire, physical design flow will be iterated a number of times until closure is achieved.

A circuit netlist is composed of a number of *components*, along with a number of *nets* that represent the required electrical connectivity between the various components, where a net connects two or more components. The chip's core layout area is comprised of a number of fixed-height *rows*. Each row consists of a number of *sites* which can be occupied by the circuit components. A *free site*

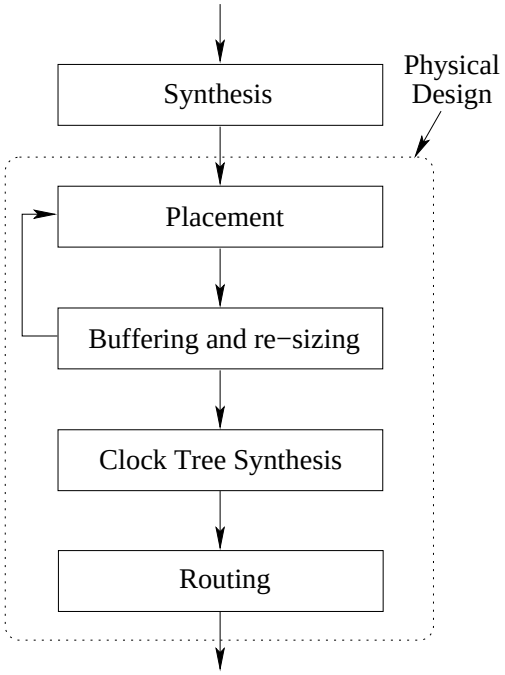


Figure I.1: A simple VLSI CMOS design flow.

is a site that is not occupied by any component. Circuit components are either *standard cells*, *macro blocks*, or *I/O pads*[2]. Standard cells have a fixed height equal to a row's height, but have variable widths. The width of a cell is an integral number of sites. On the other hand, blocks are typically larger than cells and have variable heights that can stretch a multiple number of rows. Figure I.2 gives a view of a typical placement layout. Blocks can have pre-assigned locations – say, from a previous floorplanning process – which limit the placer's task to assigning locations for just the cells. In this case, the blocks are typically referred to as *fixed blocks*. Alternatively, some or all of the blocks may not have pre-assigned locations. In this case, they have to be placed with the cells in what is commonly referred to as *mixed-mode* placement.

The major placement objectives can be summarized as follows.

[2]We will refer to standard cells by just *cells* and macro blocks by just *blocks*.

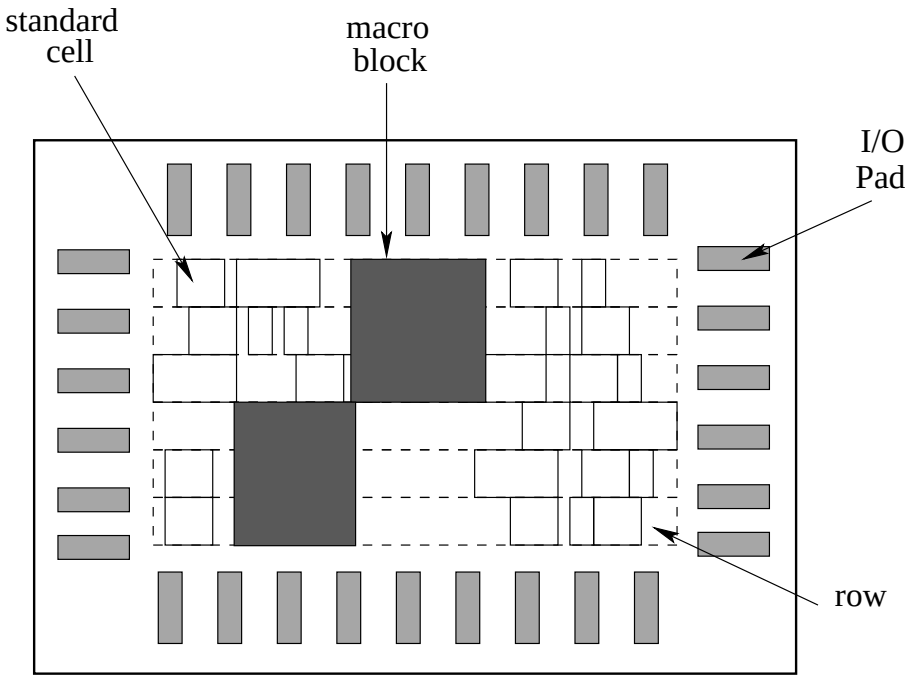


Figure I.2: The layout of a CMOS chip.

• **Wirelength.** Minimizing the total wirelength, or just wirelength, is the primary objective of most existing placers. This is no surprise given that the wirelength must be less than the fixed total routing supply of the chip, and that power and delay are proportional to the wirelength and squared wirelength respectively. Consequently, minimizing the wirelength improves the chip's performance and reduces its area. Wirelength is measured by the sum of Steiner Minimum Tree (SMT) costs of the various nets, where the SMT cost of a given point set is the minimum cost, or length, of the tree that spans all the given points as well as any additional points (Steiner points) [89]. Routed wirelength is typically slightly larger than the SMT cost since contention for routing resources by different nets might lead to detours, eventually increasing the wirelength. Given that the SMT problem is NP-hard [89], placers typically minimize and report other metrics that are faster and easier to compute. These include Half-perimeter Wirelength (HPWL) and Minimum Spanning Tree (MST). HPWL is half the perimeter of the smallest bounding box enclosing a given set of points or components. HPWL is exactly the SMT cost for

two-pin and three-pin nets, and is well-correlated to SMT cost for multi-pin (≥ 4) nets [20]. MST cost is also equal to the SMT cost for two-pin and three-pin nets, and within a constant factor for multi-pin (≥ 4) nets [89].

• **Timing.** The clock cycle of a chip is determined by the delay of its longest path, usually referred to as the *critical path*. Given a performance specification, a placer must ensure that no path exists with delay exceeding the maximum specified delay. Any excess delay over such specified value is considered *negative slack*, and timing-driven placers must minimize the magnitude of the worst negative slack and the magnitude of the total negative slack to meet performance requirements.

• **Congestion.** While minimizing total wirelength is necessary to meet the constraints of total routing resource supply, it is also necessary to meet the routing resources within various local regions of the chip's core area. A congested region might lead to excessive routing detours. Such detours can ultimately lead to excessive increase in wirelength that adversely impacts both timing and power.

• **Power.** With increasing clock frequencies and demand for battery-powered mobile devices, power is becoming an increasingly important objective to minimize. Power minimization typically involves distributing the locations of cell components to reduce the overall power consumption, alleviate hot spots, and smooth temperature gradients.

A secondary objective is placement *runtime* minimization. For a given netlist, placement is a one-time effort, and consequently it is usually tolerable to have increased runtimes if this has a positive impact on placement quality. However, for state-of-the-art designs with multi-millions of components, placement runtime can be several days, which is deemed unacceptable for fast prototyping or in timing-closure iterations. For such cases, it is important to seek methods that minimize the total placement runtime with little or no impact to placement quality [7].

I.A.1 The Wirelength Minimization Problem

We formally define the **wirelength minimization problem** as follows. We are given a circuit netlist that is represented as a hypergraph $H = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes with each node representing a circuit component, and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of hyperedges corresponding to signal nets, where a hyperedge $e_i \in E$ is a subset of nodes. The objective is to find a placement of the components on the free sites of the layout to minimize the total HPWL given by

$$z = \sum_{e_i \in E} (\max_{v_j \in e_i} x_j - \min_{v_j \in e_i} x_j + \max_{v_j \in e_i} y_j - \min_{v_j \in e_i} y_j), \tag{I.1}$$

where x_j and y_j are the vertical and horizontal coordinates of the (placed) node v_j, subject to the constraint that all nodes are placed on sites with no overlaps.

The wirelength minimization placement problem is NP-hard [74] and inapproximable [74, 73]. Solutions for the problem rely on heuristics to achieve this objective suboptimally. Given that placement is one of the oldest and first design automation problems, it has a rich history of solutions. More than four decades ago, Steinberg [81] considered placement of circuit components on a backboard such that the total wirelength is minimized. Steinberg's solution starts with an initial placement, say, obtained via random placement. Steinberg then selects an *independent set* of components, i.e., a set of components that do not share any any connections, and optimally reembeds these components within their pool of locations via optimal linear assignment. The process of selection and optimal reembedding is iterated until no further improvement in solution quality is attained.

Analytical techniques approximate the wirelength objective using quadratic [38, 21, 88, 91, 31] or nonlinear formulations [95, 61, 58, 70, 56]. The first proposal to use such methods is due to Hall [38], who suggested minimizing the squared wirelength and devised an eigenvector approach to solve the problem. Since that point, the central problems with respect to analytical techniques are how to better approximate the wirelength objective, how to numerically solve the nonlinear

objective, and how to spread out the components which typically heavily overlap in analytical solutions. Approaches for solving the nonlinear objectives include sparse matrix and conjugate-gradient methods [96]. Cell spreading techniques include the use of partitioners in top-down frameworks [21, 88, 64], network flows [91], or through the use of additional repelling forces [31, 90].

The advent of min-cut partitioners [62] paved the way to the introduction of min-cut placers [14]. The introduction of a linear-time min-cut partitioning heuristic by Fidduccia and Mattheyses [34] and terminal propagation mechanisms [30] further bolstered min-cut placement as an attractive solution. Finally, development of multi-level hypergraph partitioners [59, 8] has initiated a revival in min-cut placement [16, 92, 98, 52, 54].

Another thread of placement techniques started with the proposal of simulated annealing as a general combinatorial optimization technique [63]. In fact, placement along with the Traveling Salesman Problem (TSP) were the original two problems experimented on by Kirkpatrick, Gelatt and Vecchi [63]. Simulated annealing was quickly adopted as a leading technique in placement [77, 76], with methods such as clustering used to improve its execution time [86]. Simulated annealing can also be used with other placement methods such as min-cut to improve their performance [92].

Placement approaches typically differentiate between a *global placement* phase and a *detailed placement* phase. At the beginning of the global phase, all cells belong to one rectangular *bin* that spans the entire chip's core area. As global placement proceeds, cells are spread over the chip's core area into a number of smaller bins. By the end of this phase, each bin will typically contain only a few cells. Detailed placement is typically combinatorial in nature, and in this phase, cells are assigned exact locations and all overlaps are removed [45].

I.B Introduction to Gene Chip Placement

Gene chips – or DNA arrays – 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 [68] for a survey). The number of applications is growing at an exponential rate [36, 94], already covering a diversity of fields ranging from health care to environmental sciences and law enforcement. The reasons for this rapid acceptance of DNA arrays are a unique combination of robust manufacturing, massive parallel measurement capabilities, and highly accurate and reproducible results.

A Gene chip is simply a two-dimensional array of DNA strands (or probes). Each strand hybridizes to its Watson-Crick complement in a given RNA sample. Functionally, the array can be regarded as a way to perform massive parallel search to probe or interrogate a sample of RNA fragments for its contents. 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 [2, 35]. 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 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 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 [36, 94], 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 submicron size [68] – and the increased complexity of the design process, which leads to unpredictability of design quality and design turnaround time. Surprisingly enough, despite huge research efforts invested in DNA array applications, very few works are devoted to computer-aided optimization of DNA array design and manufacturing. Current design practices are dominated by ad-hoc heuristics incorporated in proprietary tools with unknown suboptimality. This will soon become a bottleneck for the next generation of high-density arrays.

Our work exploits the similarities between manufacturing processes for Gene Chips and VLSI CMOS chips and demonstrate significant potential for transfer of electronic design automation methodologies [32, 79, 46, 47, 50, 51, 48, 49] to the newer DNA array design field. Physical design for DNA arrays is equivalent to the physical design phase in VLSI design. It consists of two steps: *probe placement*, which is responsible for mapping selected probes onto locations on the chip, and *probe embedding*, which embeds each probe into the deposition sequence (i.e., determines synthesis steps for all nucleotides in the probe). The result of probe placement and embedding is the complete description of the reticles used to manufacture the array.

Under ideal manufacturing conditions, the functionality of a DNA array is not affected by the placement of the probes on the chip or by the probe synthesis schedule. 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 fidelity and functionality of the DNA array. 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 error 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 [2]. 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.[3] 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 [42].[4]

I.B.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 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.

Formally, the **border minimization problem** is equivalent to finding a *three-dimensional placement* of the probes [46]: two dimensions represent the site array, and the third dimension represents the nucleotide deposition sequence S (see Figure I.3). Each layer in the third dimension corresponds to a mask that induces

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

[4]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.

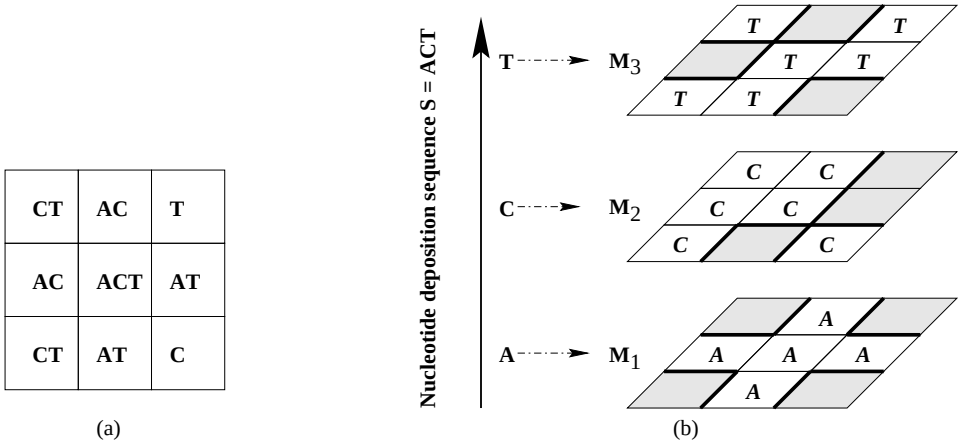


Figure I.3: (a) Two-dimensional probe placement. (b) Three-dimensional probe embedding: the nucleotide deposition sequence $S = (ACT)$ corresponds to the sequence of three masks M_1, M_2 and M_3. In each mask the masked sites are shaded and the borders between transparent and masked sites are thickened.

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.

The border minimization problem is NP-hard (it is easy to reduce the traveling salesman problem to the BMP). 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; see Figure I.4(a-b). On the other hand, *asynchronous* array synthesis permits arbitrary embeddings, as shown in Figure I.4(c-d).

In the special case of synchronous synthesis, where the embedding of a

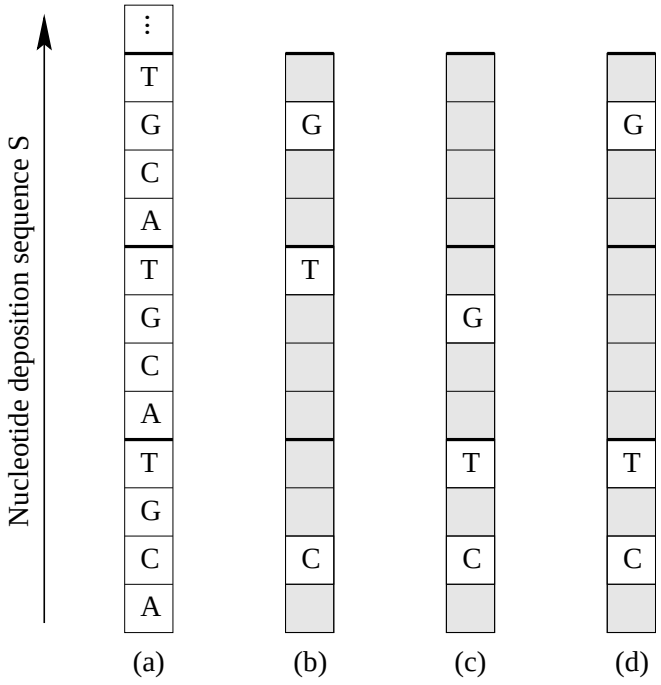


Figure I.4: (a) Periodic nucleotide deposition sequence S. (b) Synchronous embedding of probe CTG into S; the shaded sites denote the masked sites in the corresponding masks. (c-d) Two different asynchronous embeddings of the same probe.

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 twice the Hamming distance between them, i.e., twice the number of positions in which they differ.

We now develop graph-theoretical formulations for the synchronous and asynchronous variants of the array design problem. Following the development in [40], 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 the 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.[5] 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.

[5]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 (for a blank) or s_j. Thus, conflicts between two adjacent embedded probes occur only at positions where a nucleotide in one probe corresponds to a blank in the other.

I.C Organization of the Thesis and Contributions

Before embarking on improvement of placement algorithms, it is worthwhile to first ask ourselves: how much room for improvement is left with current placement algorithms? This is a *benchmarking* question and is certainly of importance to evaluate our progress – not only in placement but also in various endeavors of life. In the framework of placement and benchmarking, this thesis is concerned with two central questions: (1) Are current placement algorithms optimal (or "near" optimal) with respect to realistic CMOS and Gene chip instances?, and (2) If question (1) is answered negatively, how can the performance of placement algorithms be improved?

In light of these two questions, the organization of this thesis is simple. For each of the two technologies, CMOS chips and Gene chips, this thesis first proposes new benchmarking techniques that prove that existing placement methods display significant suboptimality (answering question (1) negatively). Then, the thesis proposes new placement methods that improve upon current algorithms and reduce the amount of suboptimality (answering question (2)). More specifically:

- Chapter II proposes a new benchmarking technique, *Zero Change Netlist Transformations (ZCNT)*, for VLSI CMOS placement. ZCNT proves for the first time that current placement algorithms display significant suboptimality behavior (up to 22%) on realistic circuits. ZCNT allows suboptimality quantification for both the HPWL and Steiner length objectives. In addition, ZCNT allows the creation of a family of similar circuits from a given seed circuit so that the consistency of placement tools can be readily evaluated.

- Given the suboptimal results proved in Chapter II, Chapter III improves the state-of-the-art of VLSI CMOS top-down placement algorithms. We propose a new technique, *placement feedback*, to improve the total placement wirelength, by giving an accurate account of the terminal propagation process in top-down placers. Implementing the proposed approach in Capo – a leading

top-down placer – leads to average improvements by 6% (and up to 15%) in wirelength.

- Chapter IV proposes three new benchmarking techniques for Gene chips by: (i) calculating lower bounds for border length, (ii) constructing chips with known optimal border length, and (iii) constructing chips with known suboptimal border length. The proposed approaches establish significant suboptimality (by up to 42%) of current Gene chip placement algorithms.

- Given the results of Chapter IV, Chapter V proposes a wealth of new techniques – constructive and iterative improvement in nature – for Gene chip placement. We propose (1) epitaxial construction placement methods, (2) scalable matching-based placement methods, and (3) partitioning-based placement methods. Furthermore, we propose probe alignment methods to further reduce the border length. Our techniques provide better results by 18% in comparison to academic arrays, and 6% better results for the Human Genome chip (U133) from Affymetrix.

- Chapter VI summarizes the main results of this thesis and points to a number of open directions.

Chapter II

New Approaches for CMOS Placement Benchmarking

Given a benchmark circuit and a placer, *placement benchmarking*, or *placer suboptimality evaluation*, is the problem of finding how close the placer's result is to the optimal result for the given benchmark. We use the term *exact suboptimality quantification* to refer to calculating the exact difference between the placer's result and the unknown optimal result, and the term *partial suboptimality quantification* to refer to calculating a lower bound on the difference between the placer's result and the unknown optimal result. The wirelength-minimization placement problem (1) is NP-hard [74], and (2) has no polynomial constant approximation algorithms [74, 73] assuming $P \neq NP$. Given these theoretical results, researchers must rely on heuristic methods to solve the problem. Lack of placement benchmarking can lead to frustration since there is no direct way to assess whether existing heuristics are sufficiently close to optimal for arbitrary instances.

In this chapter, we propose a new technique for CMOS placement benchmarking. We introduce the concept of *zero-change netlist transformations* and devise a set of such transformations to (1) exactly quantify the suboptimality of existing placers on artificially constructed instances, and more importantly (2) partially quantify their suboptimality on realistic netlists synthesized from arbitrary given netlists. Given a netlist and its placement from a placer, zero-change netlist

transformations alter the given netlist while keeping its HPWL constant, resulting in *zero change* to its HPWL. More importantly, the optimal placement HPWL of the new netlist has a value no less than the original netlist's optimal HPWL. Thus, by executing the placer on the new netlist, we can interpret any deviation of the new HPWL from the original HPWL as a lower bound on the deviation from optimal results for the new netlist. Since our transformations might change basic netlist statistics, compromising the realism of the produced netlist, we propose extensions to our approach to keep the basic netlist statistics intact. Furthermore, we also propose how to apply our technique with respect to other metrics such as rectilinear minimum spanning tree or rectilinear Steiner minimum tree costs. We support our techniques with extensive empirical results. Our results [53, 55] show that existing placers fail to reproduce their original wirelength results and incur serious deviations. The measured suboptimality gaps also increase as our transformations, as well as instance sizes, increase in magnitude.

The organization of this chapter is as follows. Section II.A classifies benchmarking approaches, summarizes previous work on the placement benchmarking problem, and gives a number of preliminaries and definitions essential to understand this work. Section II.B formally introduces the concept of zero-change netlist transformations, and their use in placement benchmarking. Section II.C gives several zero-change netlist transformations. Experimental results from the application of different transformations to various netlists are given in Section II.D. Finally, Section II.E summarizes the implications of our work and gives directions for future work.

II.A Background

Before describing our approach, we first outline the possible general benchmarking methods and previous work on the placement benchmarking problem.

Numerous heuristics have been proposed to solve the placement problem since its introduction four decades ago [81]. It is a fundamental question to ask how close a placement heuristic's (or placer's) suboptimal result is to the optimal

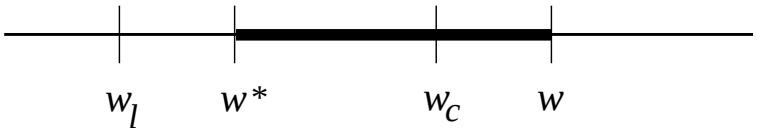


Figure II.1: Relationship between various wirelength quantities. w is a placer's suboptimal wirelength result. w^* is the optimal result. w_l is a lower bound. w_c is a pre-calculated wirelength. $w - w^*$ is the suboptimality gap.

result for a given benchmark instance. The relationship between a suboptimal result w and the optimal result w^* is schematically shown in Figure II.1. The *suboptimality gap* $w - w^*$ is an indicator of the performance of the placement heuristic relative to an optimal algorithm. Assuming that $N \neq NP$, the optimal result w^* is computationally infeasible to calculate except for extremely small instances. Thus, to estimate the suboptimality gap, it is necessary to rely on other approaches. Previous approaches can be taxonomized as follows.

- **Algorithms with Guaranteed Performance:** In this approach, an algorithm is proven to give results that do not exceed a certain upper bound on suboptimality from the optimal result. Unfortunately, for the placement problem, it has been proven that unless $P = NP$, there does not exist any polynomial-time constant ratio approximation algorithms [74, 73]. For example, Sahni and Gonzalez [74] show that if the placement problem has a polynomial-time ϵ-approximation algorithm then it is possible to decide whether or not a given graph has a Hamiltonian cycle – an NP-complete problem – in polynomial time.

- **Placement Lower Bounds:** In this approach, it would be possible to calculate a lower bound $w_l \leq w^*$ on the optimal placement of a given instance. The amount $w - w_l$ is then an upper bound on the suboptimality gap as shown in Figure II.1. Naturally, the tighter w_l is to w^*, the better is our estimate of the suboptimality gap. Placement lower bounds comprise one of the least-addressed issues in the placement literature. Donath [28] probabilisti-

cally calculated expected lower bounds for only graphs, and Donath's showed that existing placement algorithms at the time exhibit large suboptimality gaps. However, attempts by this author to replicate calculations with current placers and modern instance sizes yield extremely large suboptimality gaps. This suggests that such lower bounds are perhaps quite loose for modern instances. Direct approaches to calculate a wirelength lower bound for any given circuit using linear programming relaxations [93] have also proven unsuccessful.

- **Instances with Pre-calculated Wirelength:** In this approach, a benchmark is constructed with either *known optimal wirelength* or *pre-calculated suboptimal wirelength*. In the case of optimal constructions, a recent paper by Chang *et al.* [19] uses an overlooked construction method by Hagen *et al.* [37] to optimally construct a number of benchmarks (PEKO) with known optimal HPWL. We, however, note that this construction method has been proposed very early in the placement literature [39] more than 30 years ago. Despite [19]'s meticulous efforts to generate benchmarks with typical netlist statistics, the benchmarks are considered unrealistic since only local signals are considered. For example, a two-pin net can only be connected to spatially adjacent objects. To overcome this drawback, Cong *et al.* [23, 18] added global hyperedges and established placement upper bounds. In essence, such upper bounds are pre-calculated suboptimal wirelengths for the generated benchmarks. Another method of generating instances with known pre-calculated wirelength is through scaling a given instance and comparing the results of the placement heuristic on the scaled instance against a pre-calculated value from the unscaled instance [37]. If the pre-calculated wirelength instance is w_c then the amount $w - w_c$ is a lower bound on the suboptimality gap as shown in Figure II.1.

In addition to these approaches, a number of papers in the benchmarking literature deal with general placement methodologies or the effect of netlist

structures on the suboptimality gap. For example, results of different placers on various benchmarks are given in [4], where HPWL, timing and routability results of different placers are tabulated. The results show that placers exhibit different efficiencies on different benchmark families. Placer efficiency with respect to various netlist structures is studied by Liu and Marek-Sadowska [69]. Using the PEKO benchmark generator and existing placers, the effects of net degree distribution, net count, and Rent's exponent of the netlist are studied and tabulated. Practical conclusions are also given to justify the performance of different placers. Furthermore, Kahng and Mantik [43] study the mismatch between incremental optimizers, e.g., partitioners and ECO (Engineering-Change Order) placers, and instance perturbations. In another effort, the stability of different runs of the academic place Capo [16] on the same benchmark is studied [3], where tying a number of randomly selected cells to their regions is proposed to stabilize results from different Capo runs. Another recent paper [71] studies the reason for the suboptimal behavior of placers on the PEKO benchmarks, and concludes that poor detailed placement is the likely culprit.

II.A.1 Preliminaries

Recall that a circuit netlist is a hypergraph $H = (V, E)$, where V is a set of vertices representing the circuit cells, and E is the set of hyperedges representing the circuit wires. A hyperedge $e \in E$ is a set of vertices $e \in 2^V$, where $|e|$ gives the *cardinality* or *degree* of hyperedge e. The placement area is composed of a number of sites that cells can legally occupy. Each cell may occupy a number of sites depending on its width. A placement of a hypergraph is defined as follows.

Definition II.1. Given an enumeration of possible placement sites, a two-dimensional *placement* π of a given hypergraph $H(V, E)$ is a mapping $\pi : V \rightarrow Z^+$ assigning a placement site to every netlist cell such that no two cells overlap, i.e., no two cells share a common site. If a cell occupies more than one site then the mapping gives the first occupied site.

Definition II.2. The *Half-Perimeter Wire Length (HPWL)* of a hyperedge e in a given placement π is the length of half the perimeter of the smallest bounding box that includes all vertices of e.[1] The HPWL of a hyperedge is denoted by $l(e, \pi)$. The total HPWL (or wirelength) of a placed netlist is $L(H, \pi) = \sum_{e \in E} l(e, \pi)$.

A placement π_* of a hypergraph H that has minimum total HPWL is called an *optimal placement*. The HPWL of an optimal placement is called *the optimal wirelength or optimal HPWL*.[2] A *suboptimal placement* is a placement with a total HPWL larger than the optimal HPWL.

Definition II.3. A netlist or a hypergraph *transformation* applied to an input hypergraph $H_1 = (V, E)$ produces a new hypergraph $H_2 = (V, E')$ with the same set of vertices as H_1 but with a different set of hyperedges, i.e., a netlist transformation changes the connectivity.

With these basic definitions, we are ready to introduce the concept of zero-change netlist transformations.

II.B Suboptimality Evaluation using Zero-Change Netlist Transformations

In this section we propose and define the concept of Zero-Change Netlist Transformations (ZCNT) and examine how it can be used in placement benchmarking.

Definition II.4. Given a placement π_1 of some hypergraph H_1, a netlist transformation that synthesizes H_2 from H_1 is *zero-change* if the following two properties are satisfied:

[1] We always assume that hyperedges/nets are connected to cells via pins at the center of the cells. Another alternative is to measure half the perimeter of the bounding box completely enclosing the cells of a net.

[2] There can be more than one optimal placement yielding the same optimal HPWL.

- **Quiescency Property**: $L(H_1, \pi_1) = L(H_2, \pi_1)$, i.e., the transformation results in zero change to HPWL with respect to the input placement π_1.
- **Hardness Property**: For any other placement π_k (where $\pi_k \neq \pi_1$) : $L(H_1, \pi_k) \leq L(H_2, \pi_k)$.

The *composition* of zero-change netlist transformations is also a ZCNT. This can be proved as follows.

Theorem II.1 *The composition of ZCNTs is a ZCNT.*
Proof: If $\mathcal{Z}_j(H, \pi_1)$ denotes an arbitrary ZCNT that takes as inputs a hypergraph H and a placement π_1 and outputs a new transformed netlist, then the composition of m ZCNTs can be expressed as

$$H_2 = \mathcal{Z}_1(H_1, \pi_1), H_3 = \mathcal{Z}_2(H_2, \pi_1), \ldots, H_m = \mathcal{Z}_{m-1}(H_{m-1}, \pi_1) \tag{II.1}$$

such that for the given placement π_1,

$$L(H_1, \pi_1) = L(H_2, \pi_1) = \ldots = L(H_m, \pi_1), \tag{II.2}$$

and for any other placement π_k

$$L(H_1, \pi_k) \leq L(H_2, \pi_k) \leq \ldots \leq L(H_m, \pi_k). \tag{II.3}$$

Thus the composite transformation that produced H_k from H_1 satisfies the two required properties of ZCNTs. □

From the hardness property, it is possible to establish a relationship between the optimal placement π_1^* of the original hypergraph H_1 and the optimal placement π_2^* of hypergraph H_2 obtained from the application of one or more zero-change netlist transformations.

Theorem II.2 *Given an original hypergraph H_1, a hypergraph H_2 generated from zero-change netlist transformations applied to H_1 has an optimal HPWL no less than that of the original hypergraph, i.e., $L(H_2, \pi_2^*) \geq L(H_1, \pi_1^*)$.*

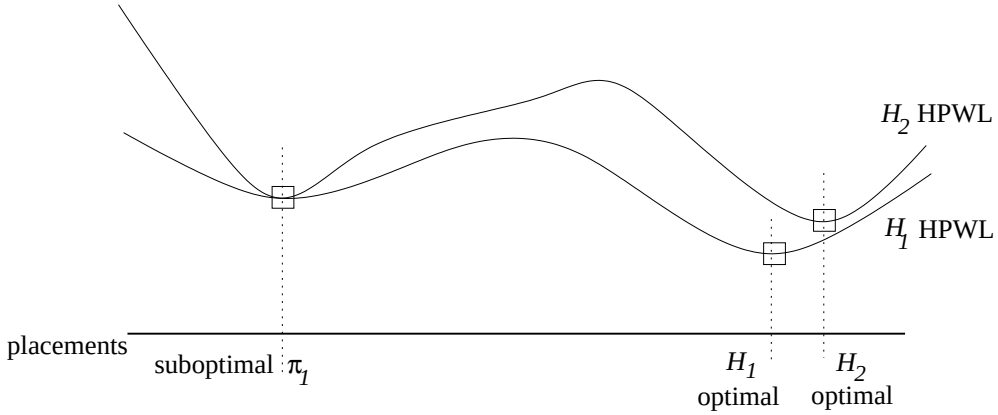


Figure II.2: A hypothetical plot showing the relationship between the placement HPWL of H_2 and H_1. The horizontal axis represent the various placements, while the vertical axis gives the HPWL values. We can see that for any placement π_k: $L(H_2, \pi_k) \geq L(H_1, \pi_k)$ and for π_1: $L(H_1, \pi_1) = L(H_2, \pi_1)$.

Proof. Towards a contradiction, assume that $L(H_2, \pi_2^*) < L(H_1, \pi_1^*)$. Using π_2^* for H_1 gives a placement with HPWL $L(H_1, \pi_2^*) \leq L(H_2, \pi_2^*)$ from the hardness property. Consequently $L(H_1, \pi_2^*) < L(H_1, \pi_1^*)$, contradicting the assumption that π_1^* is the optimal placement of H_1. □

From the previous theorem, it is easy to prove the following.

Corollary II.1 *If the given placement π_1 is optimal for H_1, i.e., $\pi_1 = \pi_1^*$, then the π_1^* is also optimal for the new hypergraph $H_2 = Z(H_1, \pi_1)$.* □

The zero-change properties as well as the results of Theorem II.2 and Corollary II.1 can be visualized using the hypothetical plot of Figure II.2.

II.B.1 Exact Suboptimality Quantification of Placers on Special Instances

Corollary II.1 leads us to a discussion of special cases where ZCNTs can be used to quantify the entire suboptimality gap. We consider a trivial netlist instance with a known optimal placement wirelength, namely, a clique $\mathcal{C}$, which has the same wirelength for any placement. We then execute ZCNTs on $\mathcal{C}$ using

any reference placement π^* (note that we can choose any random placement as the reference placement) to produce a new netlist $\mathcal{C}'$. $\mathcal{C}'$ is not trivial; nevertheless, by Corollary II.1 we have that π^* is still optimal for $\mathcal{C}'$. On the other hand, $\mathcal{C}'$ does not have the property that any arbitrary placement is optimal. If we execute a placer P on instance $\mathcal{C}'$ then the difference between the wirelength reported by P and the known optimal wirelength is *exactly* equal to the suboptimality gap of P on $\mathcal{C}'$. This method recollects methods used for testing network flow algorithms (Palubetskis' algorithm [26]), where flows are added and subtracted along certain paths such that the total flow stays the same. We will quantify the suboptimality of placers on such netlists in the experiments section below. In reality, we are interested in quantifying the suboptimality on instances that are "real" or as close to "real" as possible. Thus, we next examine how to use ZCNTs to calculate *lower bounds* – instead of exact bounds as in the case of the clique – on the suboptimality gaps of netlists that have been synthesized from general netlists.

II.B.2 Partial Suboptimality Quantification of Instances Synthesized from General Instances

The use of zero-change netlist transformations for general benchmarking is illustrated in Figure II.3. Given a netlist H, placer P produces a placement π_1 with HPWL $w_1 = L(H_1, \pi_1)$. Given π_1, applying zero-change netlist transformations to H_1 produces a new netlist H_2. From the quiescency property, $L(H_2, \pi_1) = L(H_1, \pi_1)$. However, executing P on H_2 produces a new placement π_2 with some wirelength $w_2 = L(H_2, \pi_2)$. The main question is whether $w_1 = w_2$. If the placer is optimal then $w_1 = w_2$. If the placer is suboptimal then there are three possibilities:

1. $w_1 = w_2$ indicating that the placer is stable and not sensitive to the netlist transformations.

2. $w_2 < w_1$ indicating that the original placement was not optimal for the original netlist H_1 and that the transformations lead the placer to a better suboptimal

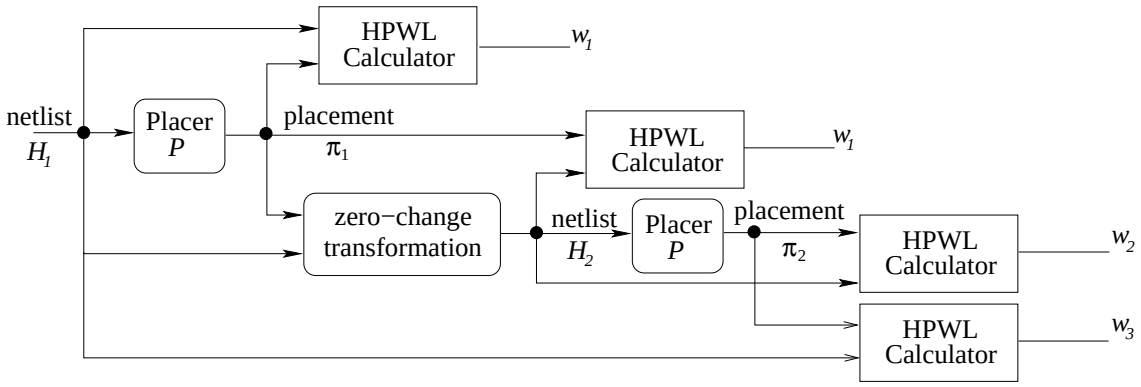


Figure II.3: Conceptual presentation of zero-change netlist transformations. The difference $w_2 - w_1$ gives a measure of suboptimality for placer P.

placement π_2 for H_1 since $L(H_1, \pi_2) \leq w_2 = L(H_2, \pi_2) < w_1 = L(H_2, \pi_1) = L(H_1, \pi_1)$.

3. $w_2 > w_1$ showing that the placer is suboptimal and sensitive to the netlist transformations. Since w_1 acts as an upper bound to the optimal placement of H_2, we have that $w_2 - w_1 = L(H_2, \pi_2) - L(H_2, \pi_1)$ is a lower bound on the suboptimality gap of placer P on the new netlist H_2, which is equal to $w_2 - L(H_2, \pi_2^*) = L(H_2, \pi_2) - L(H_2, \pi_2^*)$.

An important characteristic of zero-change netlist transformations is that the optimal placement HPWL of the new netlist H_2 is no less than that of the original netlist H_1, as we established in Theorem II.2. Thus, executing the placer on the new netlist is likely to yield the third possibility, where $w_2 > w_1$. This will be empirically demonstrated in Section II.D. The possibility of using zero-change netlist transformations to calculate lower bounds on the suboptimality gaps of the synthesized netlists raises a number of interesting questions:

- *How tight is the calculated lower bound?* Answering this question entails calculating the exact suboptimality gap for the synthesized netlist which is as hard as calculating the suboptimality gap for the original netlist. Thus, the calcu-

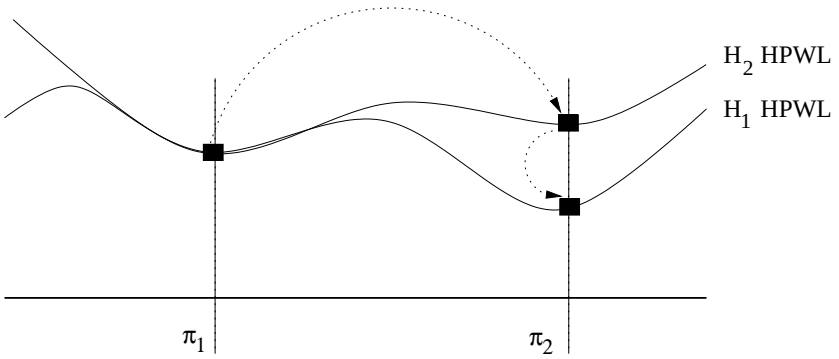


Figure II.4: Transformations can improve placer performance.

lated lower bound can only serve as a certification of placer suboptimality on the synthesized netlist by at least its value.

- *Can the suboptimality gap on a synthesized netlist reveal any information about the suboptimality gap on its original netlist?* This is unlikely. However, we can regard the deviation in wirelength as a form of placer sensitivity to transformations applied to the original netlist.

One may wonder about $w_3 = L(H_1, \pi_2)$ which is produced using π_2 for the original netlist H_1. This raises the possibility of using netlist transformations to improve the placeability of netlists. This concept is illustrated in Figure II.4, where dotted arrows represent the chain of events. First, H_2 is obtained from H_1 using π_1. By definition of ZCNT, $L(H_2, \pi_1) = L(H_1, \pi_1)$. Second, the placer is executed on H_2 which leads to another local suboptimal point π_2 with HPWL $L(H_2, \pi_2)$. Finally, the netlist H_2 is discarded, but nevertheless the placement π_2 is used for the original hypergraph H_1 with a HPWL of $L(H_1, \pi_2)$. Note that from the hardness property, $L(H_1, \pi_2) \leq L(H_2, \pi_2)$. If $L(H_1, \pi_2) < L(H_1, \pi_1)$ then the transformations have improved the performance of the placer.

From the taxonomy introduced in the previous section, it is clear that our zero-change methodology fits in the category of instance generation with precalculated wirelength. One key difference between our approach and other approaches is the ability to extract partial suboptimality information from synthe-

sized netlists for any given arbitrary benchmark. We now propose a number of zero-change netlist transformations to assess the performance of different placers.

II.C Zero-Change Netlist Transformations

In this section we give a number of netlist transformations, which change the netlist connectivity but not the vertex set, and which have the key properties of zero-change netlist transformations. We propose three basic transformations: hyperedge cardinality increase, hyperedge decomposition, and edge substitution. We extend the applicability of these transformations in two ways: first, we compose them in a *hybrid* fashion, and second, we embed them in a flow that preserves basic netlist statistics. Finally, we discuss how to evaluate placer suboptimality using other metrics such as RMST and RSMT. We start by introducing the hyperedge cardinality increase transformation.

II.C.1 Hyperedge Cardinality Increase

The purpose of this transformation is to assess the sensitivity of placers to hyperedge cardinality increase by examining the impact of increasing the cardinality of hyperedges. We only increase the cardinality of hyperedges of degree ≥ 3. Our transformation is simple: given a netlist H and its placement π_1, the bounding box of each hyperedge (excluding 2-pin edges) is calculated, and an additional number of vertices is added to each hyperedge from within its bounding box. The cardinality increase procedure HYPERC is given in Figure II.5. In our experiments, we limit the amount of hyperedge cardinality increase, and if there are a number of vertices inside the bounding box to choose from, we always prioritize vertices of the least degree to break the ties. Before proving that procedure HYPERC is a ZCNT, we first state the following lemma which is easy to prove.

Lemma II.1 *Given two sets of nodes S_1 and S_2: if $S_1 \subseteq S_2$ then $l(S_1, \pi_k) \leq l(S_2, \pi_k)$ for any placement π_k.* □

Lemma II.1 basically states that HPWL is *monotone* [13].

Theorem II.3 *Procedure HYPERC in Figure II.5 is a zero-change netlist transformation.*

Proof: If the netlist produced by procedure HYPERC has the quiescency and hardness properties of Definition II.4 then the theorem is proved. We will prove that each of these properties holds.

- Quiescency: Given a hypergraph H_1 and a placement π_1, applying procedure HYPERC produces a new hypergraph H_2. By construction, adding a number of vertices to a hyperedge from within its bounding box does not change its HPWL value. Therefore, $L(H_2, \pi_1) = L(H_1, \pi_1)$.

- Hardness: Given some $\pi_k \neq \pi_1$, H_1 would have an HPWL value of $L(H_1, \pi_k)$. Replacing each hyperedge e_i in H_1 with e_i' according to procedure HYPERC yields a HPWL value of $L(H_1, \pi_k) + \sum_i (l(e_i', \pi_k) - l(e_i, \pi_k)) \geq L(H_1, \pi_k)$ since $l(e_i', \pi_k) \geq l(e_i, \pi_k)$ from Lemma II.1. Thus $L(H_2, \pi_k) \geq L(H_1, \pi_k)$. □

Finally, we note that the inverse or "anti" transformation to HYPERC, where a hyperedge's cardinality is decreased, does not satisfy the zero-change requirements since it might yield a netlist with lower optimal HPWL than the original netlist.

II.C.2 Hyperedge Decomposition

Our second transformation simplifies a hyperedge by *decomposing*, or partitioning, it into two intersecting hyperedges, with each of the new hyperedges having smaller cardinality than the original hyperedge. We define an optimal hyperedge decomposition as follows.

Definition II.5. A hyperedge e is *optimally decomposable* in some placement π_k if it is possible to decompose e into two intersecting hyperedges e_1 and e_2 such that

Input: A hypergraph $H_1 = (V, E_1)$ and a placement π_1 for H.
Output: A new hypergraph $H_2 = (V, E_1)$.

1. Initialize $E_2 = E_1$.
2. For each hyperedge $e_i \in E_2$ where $|e_i| \geq 3$:
   3. Find the set of vertices C_{e_i} enclosed within the bounding box of e_i in placement π_1.
   4. If $C_{e_i} \neq \emptyset$ then augment hyperedge e_i as follows: $e_i = e_i \cup S_e$, where $S_{e_i} \subseteq C_{e_i}$.
5. Return hypergraph $H_2 = (V, E_2)$.

Figure II.5: Procedure HYPERC for hyperedge cardinality increase.

$l(e, \pi_k) = l(e_1, \pi_k) + l(e_2, \pi_k)$ and $e = e_1 \cup e_2$.

Figure II.6 shows a hyperedge, whose bounding box is represented by a dashed line, optimally decomposed into two hyperedges as shown by the dashed rectangles.

Lemma II.2 *For any two hyperedges e_i and e_j with $|e_i \cap e_j| \neq \emptyset$, $\max(l(e_i, \pi_k), l(e_j, \pi_k)) \leq l(e_i \cup e_j, \pi_k) \leq l(e_i, \pi_k) + l(e_j, \pi_k)$ in any placement π_k.*
Proof: The proof is by enumerating all possible cases for bounding boxes of e_i and e_j. Since $|e_i \cap e_j| \neq \emptyset$, there are only three possible configurations for the bounding boxes of e_i and e_j.

1. *Contained:* In this case, the bounding box of one hyperedge is completely contained within the other bounding box as shown in Figure II.7.a. In this case $l(e_i \cup e_j, \pi_k) = \max(l(e_i, \pi_k), l(e_j, \pi_k))$.

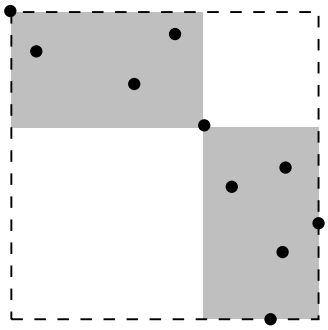


Figure II.6: An example of an optimally decomposable hyperedge.

2. *Overlapping:* In this case, the two bounding boxes overlap with $l(e_i \cup e_j, \pi_k) < l(e_i, \pi_k) + l(e_j, \pi_k)$ as shown in Figures II.7.b and II.7.c.

3. *Touching:* In this case, the two bounding boxes touch each other at a common vertex with $l(e_i \cup e_j, \pi_k) = l(e_i, \pi_k) + l(e_j, \pi_k)$ as shown in Figure II.7.d. □

From Lemma II.2, it is easy to see the following.

Lemma II.3 *e_1 and e_2 give an optimal decomposition of e in some placement π_k only if $|e_1 \cap e_2| = 1$ and the bounding boxes of e_1 and e_2 touch at their common vertex.* □

Lemma II.3 provides us with a simple characterization to optimally decompose any hyperedge. Using this characterization, we devise a procedure for optimal hyperedge decomposition (procedure HYPERD), given in Figure II.8. The procedure examines every hyperedge with degree larger than two, and checks whether it is possible to optimally decompose it at each of its vertices. If there exists more than one possible vertex at which the optimal decomposition can occur, we break the tie by choosing the vertex that results in the most balanced decomposition or partition. We next prove that procedure HYPERD is a zero-transformation procedure.

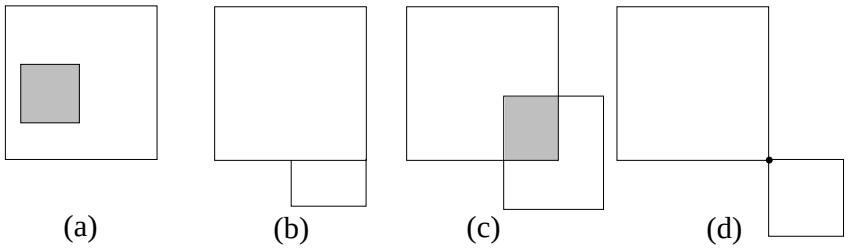


Figure II.7: Enumeration of possible bounding box configurations.

Theorem II.4 *Procedure HYPERD is a zero-change netlist transformation.*

Proof: If the netlist produced by procedure HYPERD has the quiescency and hardness properties of Definition II.4 then the theorem is proved. We will prove that each of these properties holds.

- Quiescency: Since procedure HYPERD decomposes edges only optimally according to Lemma II.3, $L(H_1, \pi_1) = L(H_2, \pi_1)$.
- Hardness: Given some $\pi_k \neq \pi_1$, H_1 would have an HPWL value of $L(H_1, \pi_k)$. Replacing each hyperedge e_i in H_1 with e_i' to yield H_2 according to procedure HYPERD yields a HPWL value of $L(H_1, \pi_k) + \sum_i (l(e_i^1, \pi_k) + l(e_i^2, \pi_k) - l(e_i, \pi_k)) \geq L(H_1, \pi_k)$ since $l(e_i^1, \pi_k) + l(e_i^2, \pi_k) \geq l(e_i, \pi_k)$ by Lemma II.2. Thus $L(H_2, \pi_k) \geq L(H_1, \pi_k)$. □

Before ending this subsection, we note that the inverse or "anti" transformation to HYPERD, i.e., merging two touching hyperedges into one bigger hyperedge, does not satisfy the zero-change requirements since it might yield a netlist with lower optimal HPWL than the original netlist.

II.C.3 Edge Substitution

Our third transformation deals with edges, i.e., hyperedges of degree two. Our transformation does not change the cardinality of edges but rather increases the total number of two-pin edges. We start with the following fact that charac-

Input: A hypergraph $H_1 = (V, E_1)$ and its placement permutation π_1.
Output: A hypergraph $H_2 = (V, E_2)$.

1. Iterate until there is no possible decomposition:
2. Set $E_2 = \emptyset$.
    3. For each hyperedge e_i in E_1:
        4. For each vertex $v_j \in e_i$:
            5. If e_i can be partitioned into two sets e_i^1 and e_i^2 such that the bounding boxes of e_i^1 and e_i^2 touch each other at v_j
            then insert e_i^1 and e_i^2 into E_2 and goto Step 3.
            else insert e_i into E_2.
    6. Set $E_1 = E_2$.
7. Return hypergraph $H_2 = (V, E_2)$.

Figure II.8: Procedure HYPERD for hyperedge decomposition.

terizes the triangle inequality in Manhattan or rectilinear metric.

Fact II.1. If d_{ij} gives the Manhattan distance between two placement sites i and j then $d_{ij} \leq d_{iq} + d_{qj}$ for any site q, and if q lies within the bounding box defined by sites i and j then $d_{ij} = d_{iq} + d_{qj}$.

We leverage Fact II.1 for the netlist transformation as given in procedure EDGESUB of Figure II.9, where we take every edge, calculate its bounding box, and substitute it with two edges by using a third vertex from within its bounding box. Since there can be more than one vertex that is eligible to take the role of the third vertex, we always break ties in favor of the vertex of the least degree. Edge substitution can be carried out more than once, effectively transforming an

Input: A hypergraph $H_1 = (V, E_1)$ and its placement π_1.
Output: A hypergraph $H_2 = (V, E_2)$.

1. Initialize $E_2 = E_1$.
2. Find a two-pin edge $\{u, v\}$ in E_2 and calculate its bounding box in π_1.
3. Find a node p inside the bounding box of $\{u, v\}$.
4. If such a node p exists then
    5. Delete $\{u, v\}$ from E_2 and insert two new two-pin edges $\{u, p\}$ and $\{p, v\}$ in E_2, i.e., $E_2 = E_2 \setminus \{\{u, v\}\} \cup \{\{u, p\}, \{p, v\}\}$.

Figure II.9: Procedure EDGESUB for two-pin edge substitution.

edge between sites i and j into a path between sites i and j as depicted in Figure II.10.

Theorem II.5 *Procedure EDGESUB is a zero-change netlist transformation.*
Proof: If the netlist produced by procedure HYPERD has the quiescency and hardness properties of Definition II.4 then the theorem is proved. We will prove that each of these properties holds.

- Quiescency: Given a hypergraph H_1 and a placement π_1, applying procedure EDGESUB produces a new hypergraph H_2. By construction, substituting an edge by two edges, or more, using nodes from within its bounding box does not change the total HPWL.
- Hardness: Given some $\pi_k \neq \pi_1$, H_1 has HPWL value of $L(H_1, \pi_k)$. Substituting every edge $\{u_i, v_i\}$ by two edges $\{u_i, p_i\}$ and $\{p_i, v_i\}$ gives a new netlist H_2 such that $L(H_2, \pi_k) = L(H_1, \pi_k) + \sum_{\forall \{u_i, v_i\}} (l(\{u_i, p_i\}, \pi_k) + l(\{p_i, v_i\}, \pi_k) - l(\{u_i, v_i\}, \pi_k)) \geq L(H_1, \pi_k)$ by Fact II.1.

□

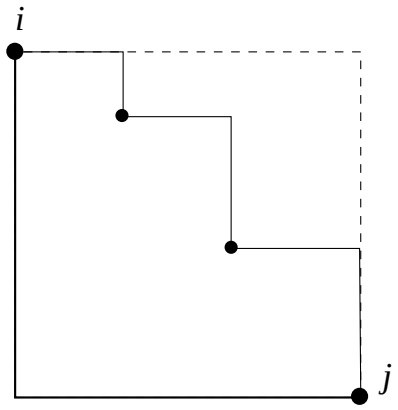


Figure II.10: Substituting an edge between i and j by a path according to procedure EDGESUB does not change the wirelength of a given placement.

We note that if procedure HYPERC of Subsection II.C.1 (for hyperedge cardinality increase) is allowed to operate on two-pin edges, then transformation EDGESUB can be considered as a combination of hyperedge cardinality increase (HYPERC) on two-pin nets, immediately followed by the hyperedge decomposition (HYPERD) of Subsection II.C.2.

As a final remark: the inverse or "anti" transformation to EDGESUB, by substituting a path of edges with a single edge, does not satisfy the zero-change requirements.

II.C.4 Hybrid Transformations

It is possible to apply the previous three transformations on a given netlist and empirically examine the collective impact of all transformations. As proved in Section II.B, the hybrid or composite application of all zero-change netlist transformations is also zero-change. For example, we can compose hyperedge cardinality increase, hyperedge decomposition, and edge substitution in sequence to yield a *hybrid zero-change netlist transformation.*

II.C.5 Transformations Preserving Netlist Statistics

All of our zero-change netlist transformations increase a certain characteristic of the netlist, such as hyperedge cardinality or total number of hyperedges. We tabulate the impact of our transformations on basic netlist statistics in Table II.1. The table gives the increase in number of hyperedges and total pin count, i.e., total hyperedge cardinality, for each one of our transformations. Since circuit instances have a ratio of about one between the number of nets and number of cells, our previous transformations synthesize netlists with a larger ratio. It is certainly desirable that the new hypergraph instance has the same statistics as the original hypergraph. Thus, we investigate in this subsection how to produce a new hypergraph with reasonable (e.g., preserving realism) statistics.

A number of papers research what constitutes a realistic netlist [66, 84, 82]. For example, realistic netlists exhibit typical values for (1) the number of hyperedges in comparison to the nodes, (2) the average node degree, (3) a hyperedge cardinality power-law distribution [83], and (4) Rent parameter [66].

To keep a degree of realism in our generated hypergraphs, we use a simple strategy: before the initial placement, we pre-process the input hypergraph to reduce its hyperedge cardinality and number of hyperedges by exactly the amount that will be increased due to zero-change netlist transformations. Using this strategy, a given input hypergraph H_1 is pre-processed to a new hypergraph H_1' which is then used to derive the ZCNT flow of Figure II.3. The final hypergraph H_2 pro-

Table II.1: Impact of ZCNTs on basic netlist statistics.

Transformation	Netlist Characteristic	
	Number of hyperedges	Total pin count
Hyperedge Cardinality Increase	0	+1
Hyperedge Decomposition	+1	+1
Edge Substitution	+1	+2

duced from applying zero-change netlist transformations to H_1' will have the same amount of hyperedges and total hyperedge cardinality as the original hypergraph H_1.

Certainly such a framework of hypergraph pre-processing followed by application of zero-change netlist transformations does not produce a realistic netlist; nevertheless, it keeps the "basic" netlist statistics intact. Possible pre-processing steps include:

- Removal of random hyperedges to decrease the number of hyperedges in a netlist.
- Converting multiple hyperedges into one hyperedge. For example, given a number of hyperedges $e_1, e_2, \ldots, e_k$, we can convert them into one bigger hyperedge by deleting all of them and creating a new hyperedge $e_1 \cup e_2 \cup \ldots \cup e_k$.
- Reduction of hyperedge cardinality by removing an arbitrary node from any hyperedge with three or more nodes.

II.C.6 Benchmarking Other Metrics: RMST and RSMT

Rectilinear Steiner Minimum Trees (RSMT) or Rectilinear Minimum Spanning Trees (RMST) may be contrasted with HPWL in a number of ways.

- The bounding box of a net in a given placement is unique; however, there might be more than one minimum RMST or RSMT. Furthermore, since RSMT is an NP-hard problem [89], we can only approximate it using a heuristic (suboptimal) RSMT. There are numerous heuristic RSMT constructions, and one possibility is to use the RMST as a possible RSMT heuristic with a performance ratio of at most $\frac{3}{2}$ [89]. In practice, the average RMST/RSMT ratio for a uniformly random sets of n points chosen from the unit square approaches 1.12 [57].
- HPWL is *monotone*. Given any placement π_k: if $S_1 \subseteq S$ then $L(S_1, \pi_k) \leq L(S, \pi_k)$. The monotone property also holds for Steiner minimum tree (SMT) since augmenting a set of points by an additional point cannot reduce the length

of the SMT connecting these points. On the other hand, such a property does not hold for the minimum spanning tree: adding an additional point to an existing point set might actually reduce the length of the MST connecting these points. This is the principle underlying the Steiner tree concept [13, 80].

In this work, we use the RMST as a heuristic to construct the RSMT as suggested by Property 1. Thus, we focus our discussions on the RMST. Given a set of nodes S representing a hyperedge, and a placement π_1, our transformations for RMST/RSMT suboptimality evaluation are as follows.

- Hyperedge cardinality increase: According to our previous discussion, adding a node from within the bounding box of a net S might either increase or decrease the RMST length of S. Thus, we modify the hyperedge cardinality in a simple way to handle this: after increasing the cardinality, we re-calculate the RMST value of all nets and use the total RMST as the pre-calculated RMST value of the netlist. This value is used to benchmark the RMST value produced when the placer is executed on the new netlist.[3]

- Hyperedge decomposition: If hyperedge S is optimally decomposed into two hyperedges S_1 and S_2 then the RMST cost of S_1 plus the RMST cost of S_2 is greater than or equal to the RMST cost of S. This can be proven by contradiction; if the RMST cost of S_1 plus the RMST cost of S_2 is less than the RMST cost of S, then we can "concatenate" the RMSTs of S_1 and S_2 to obtain a new RMST for S that has less cost than the original minimal RMST of S.

- Edge substitution: In this case the RMST is equivalent to HPWL since this transformation only applies to two-pin edges and produces two-pin edges where both RMST and HPWL have the same value.

[3]Since the RMST length might change after the transformation, the transformation is no longer zero-change. In the "lucky" case where the final RMST value is higher than the pre-calculated value, the lower bound on suboptimality, as measured by the difference between the new and precalculated values, is likely to be less than those corresponding to ZCNTs.

Our discussion of various netlist transformations is now complete. We empirically explore the impact of zero-change netlist transformations in the next section.

II.D Experimental Results

In this section we empirically evaluate the suboptimality of existing placers with respect to our proposed transformations. This evaluation is carried out using the IBM benchmarks (version 1)[4] and four academic placers. The components of the IBM (version 1) benchmarks are comprised of standard cells with varying widths, with all pins placed by default at the center of their respective cells. We use the following placers in our empirical evaluation.

- Capo [16] (version 9.0 with feedback [52]): Capo uses a min-cut engine in a top-down bisection framework to deliver fast results.
- FengShui [98] (version 2.6): FengShui uses a min-cut engine based on iterative deletion in a top-down framework.
- Dragon [92] (version 3.01): Dragon uses a min-cut engine in a top-down quadrisection framework. Dragon also uses simulated annealing to improve its results.
- mPL (version 4.0) [24]): mPL is an analytical placer that uses a non-linear programming formulation in a multi-level optimization framework.

Since all pins are placed at the center of their respective cells in all circuits of the IBM benchmarks, we measure HPWL center-to-center, and any additional pins necessitated by our transformations are also placed at the centers of cells. Before carrying out our experiments, we estimate the noise [3, 44] of different placers by reporting the average difference in HPWL for two different executions of a single placer on the same netlist, but with different ordering of nets [43]. Our

[4]Other benchmarks like the PEKO instances [19] are not suitable since all of their nets are local in the optimal placement. This leaves no room to exploit our transformations. For example, there is no possibility of executing two-pin edge substitution.

Table II.2: Statistics of special netlists obtained by applying ZCNTs to cliques.

bench-	Statistics after ZCNT		
mark	nodes	nets	pins
C100	100	6435	15480
C400	400	103740	207480
C900	900	525915	1051830
C1600	1600	1662960	3325920

results show that FengShui has a noise margin of around 0.92%, mPL has a noise margin of around 0.89%, Capo has a noise margin of around 2.9%, and Dragon has a noise margin of around 3.37%.

II.D.1 Using ZCNT to Quantify the Exact Suboptimality Gap of Placers on Special Instances

In this special case, we first construct a clique and fix any placement as its optimal reference placement (we use square layouts). This is valid since all placements give the same optimal wirelength for a given clique. Using the reference placement, we transform the clique using ZCNTs to a non-trivial instance, by increasing the total hyperedge cardinality by 30%, and increasing the total number of two-pin edges by 30% using edge substitution. We then execute the placers on the new netlist and report the difference between the observed wirelength and the known optimal wirelength. Table II.2 gives the statistics of netlists generated by applying ZCNTs to the cliques, and we give the suboptimality results in Table II.3. We observe that the suboptimality gap is small and around 1-3%. Certainly, such instances do not resemble typical VLSI instances, but they nevertheless demonstrate how ZCNTs can be used to exactly quantify the suboptimality gap.

Table II.3: Exact suboptimality quantification for special instances. A '-' indicates that the placer failed while placing the instance. We report the actual HPWL and the percentage deviation from the optimal placement between parentheses.

bench-	Optimal	Placer			
mark	HPWL	Capo	FS	Dragon	mPL
C100	33000	33301 (0.91%)	-	33502 (1.52%)	33086 (0.26%)
C400	1064000	1085590 (1.97%)	-	-	1071430 (0.65%)
C900	8091000	8367070 (3.33%)	-	-	-
C1600	34112000	35019800 (2.64%)	-	-	-

II.D.2 Using ZCNTs to Partially Quantify the Suboptimality Gap of Placers on Instances Synthesized from General Instances

In this subsection, we use ZCNTs to partially quantify placers suboptimality, i.e., by calculating lower bounds on the suboptimality gap of the placers on instances synthesized from general instances. Our experimental execution flow is based on the outline of Figure II.3. We note that before applying our transformations, we sort all nets by their HPWL in the given placement in a decreasing order. In all experiments, we report the percentage deviations $\frac{w_2-w_1}{w_1}$, and in only one experiment, HYPEREDGE DECOMPOSITION, we report $\frac{w_3-w_1}{w_1}$. The first amount, $\frac{w_2-w_1}{w_1}$, is a lower bound on deviation from the optimal HPWL of the new netlists. The second amount, $\frac{w_3-w_1}{w_1}$, is reported to see if the transformations can lead to an improvement in the placeability of the original netlists. We have encountered a limited number of unsuccessful placement executions. Such runs are designated in the Tables by a dash $(-)$.

Hyperedge cardinality increase

In a first series of experiments, we empirically determine the performance of placers with respect to the zero-change hyperedge cardinality increase transformation as given by procedure HYPERC. Table II.4 gives the results of HYPERC with a total cardinality, i.e., total pin count, increase of 20% for each benchmark. We report the deviation of each placer HPWL with respect to its own placement. From the results, it is clear that none of the four placers manages to maintain their original HPWL. All placers exhibit a substantial unnecessary increase in HPWL.

To further study the impact of zero-change hyperedge cardinality increase, we focus on the IBM01 benchmark and measure the HPWL in response to a total cardinality increase from 0 to 25% in increments of 5%. We plot the results in Figure II.11. Notice that in contrast to Table II.4, we directly report the HPWL values and not the percentage deviations. From the figure, current placers exhibit a suboptimal behavior, and there is a general trend of increasing HPWL in response to hyperedge cardinality. We also notice that the relative performance ranking of various placers differs depending on the amount of hyperedge cardinality increase. For example, FengShui's performance deteriorates more gracefully than Dragon's.

Hyperedge decomposition

In a second series of experiments, we empirically determine the performance of placers to zero-change hyperedge decomposition as given by the HYPERD procedure. All hyperedges are decomposed until no further decomposition is possible. The empirical results are given in Table II.5, where the percentage changes $\frac{w_2-w_1}{w_1}$ and $\frac{w_3-w_1}{w_1}$ are reported. It is clear from the empirical results that w_3 is always less than w_2 in agreement with the *hardness* property. We also notice that placers are sometimes able to exploit such transformations to slightly improve their results.

We also carry out a more detailed study on the ibm01 benchmark, shown in Figure II.12. In the plot, the x-axis gives the amount of zero-change hyperedge

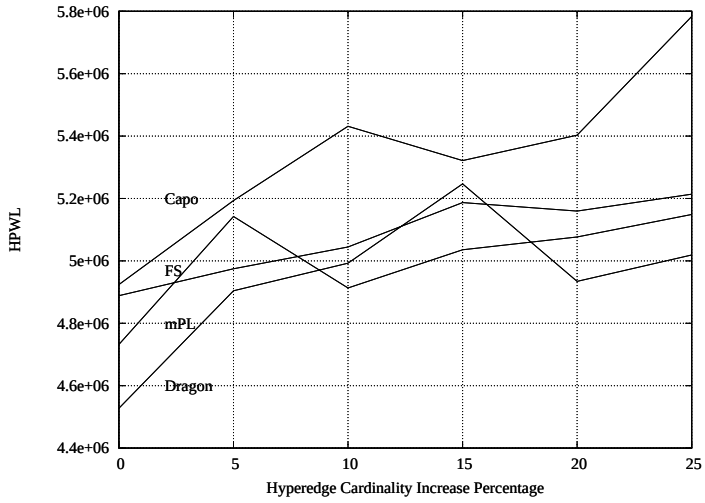


Figure II.11: Effect of zero-change total hyperedge cardinality (total pin count) increase on the performance of various placers on the ibm01 benchmark. Total hyperedge cardinality increase is varied from 0 to 25% in increments of 5%.

decomposition in increments of 5%, and the y-axis gives the HPWL produced from the various placers. In general, the response of placers to hyperedge decomposition is relatively "milder" than the response to hyperedge cardinality increase in Figure II.11. The magnitude of changes in HPWL is relatively small, and the performance is overall quite stable.

From the results, it is not clear that hyperedge decomposition can improve the performance of existing placers. Zero-change hyperedge decomposition simplifies a netlist by taking large hyperedges and optimally decomposing them into two or more smaller hyperedges. This simultaneously decreases the cardinality of hyperedges and increases the number of hyperedges. Our results show no improvement in performance due to hyperedge decomposition. We can envision that perhaps by selective hyperedge decomposition and careful tuning, this transformation can be used within a placement run to simplify netlists and lead to better placements.

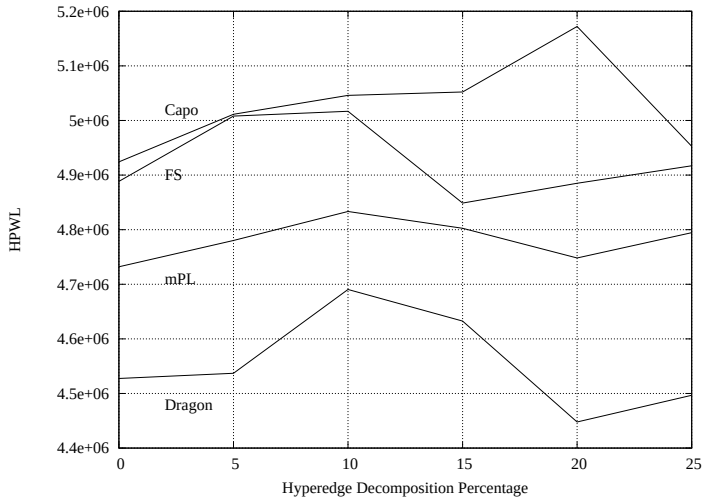


Figure II.12: Effect of zero-change hyperedge decomposition on the performance of various placers on the ibm01 benchmark. The amount of hyperedge decomposition is varied from 0 to 25% in increments of 5%.

Edge substitution

In a third series of experiments, we determine the performance of existing placers with respect to zero-change edge substitutions using procedure EDGESUB. We present our results in Table II.6, where we keep substituting edges until the number of nets increases by 10%. The results show that placers exhibit a systematic unnecessary increase in HPWL. To further study the impact of procedure EDGESUB, we apply it to the ibm01 benchmark in varying amounts, increasing the total number of nets from 0 to 25% in increments of 5%. The HPWL results are plotted in Figure II.13. From the plot, we notice that placers exhibit a consistent increase in HPWL; the larger the amount of substitution, the greater the amount of HPWL. Overall, we conclude that zero-change edge substitutions lead placers to display suboptimal behavior.

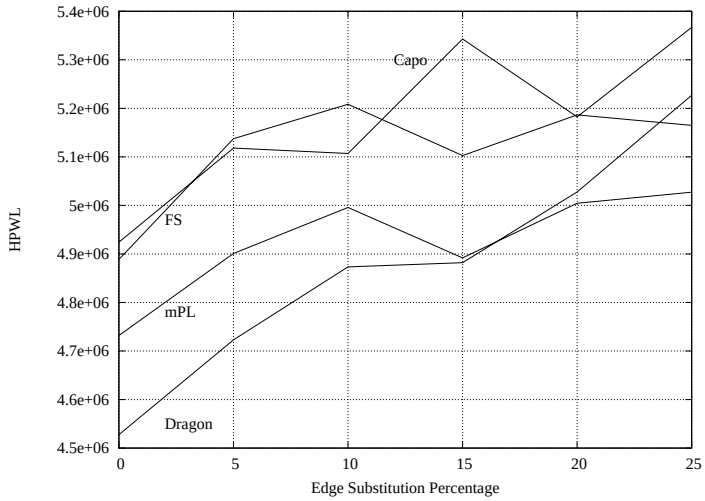


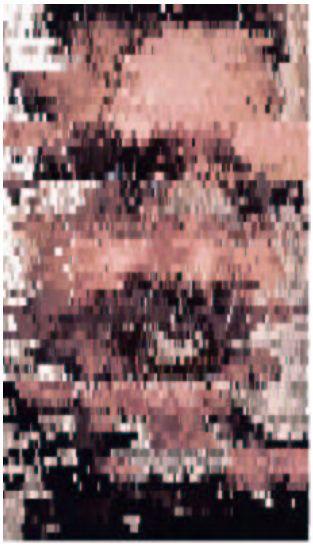
Figure II.13: Effect of zero-change edge substitution on the performance of various placers on the ibm01 benchmark. The amount of hyperedge decomposition is varied from 0 to 25% in increments of 5%.

Hybrid transformations

In a fourth series of experiments, we test the performance of placers with respect to hybrid transformations. We give our results in Table II.7. Given the initial placements of the various placers on different benchmarks, we apply (1) hyperpedge cardinality increase leading to a total cardinality increase of 5%, (2) edge substitution increasing the total number of nets by 5%, and (3) hyperedge decomposition further increasing the total number of nets by an additional 5%. The results show that placers again exhibit suboptimal behavior with respect to hybrid zero-change netlist transformations.

Finally, we visually trace the the discrepancy between the reference placement and the second new placement by using photos [71]. We impose a photo on the reference placement as shown in Figure II.14.a, with each cell receiving a piece of the photo corresponding to its location in the placement. Then, we visualize the differences by printing the new placement with the photo pieces still on top of the cells, as given in Figure II.14.b.

(a) A photo imposed on the reference placement. HPWL = 4.97.



(b) The photo as dictated by the new placement. HPWL = 5.71.

Figure II.14: Tracing suboptimality of Capo9.3 on the ibm01 benchmark. The application of ZCNTs reveals a suboptimality gap of at least 15.82%.

Transformations preserving netlist statistics

In this experiment, we apply the techniques of Subsection II.C.5 to partially evaluate placer suboptimality while keeping a degree of realism in the transformed netlists. We start by pre-processing all netlists to reduce the absolute amounts of hyperedges and total hyperedge cardinality by k and $2k$, respectively, where $k \geq 1$. This can be achieved by applying the following k times:

- Join any two hyperedges that share a common node into one larger hyperedge. This reduces both the number of hyperedges and the total hyperedge cardinality (or total pin count) by one.
- Reduce the hyperedge cardinality by removing a vertex from any arbitrary hy-

peredge that has degree greater than or equal to three.

The pre-processed netlists are placed and zero-change edge substitutions are then applied k times to yield a transformed netlist with the same amount of total nets and total hyperedge cardinality as the original netlist. The new netlist thus has the same basic netlist statistics as the original,even though the netlists differ structurally. HPWL suboptimality deviations of the transformed netlists are reported in Table II.8, where k is chosen to be 10% of the number of nets in each netlist. The results demonstrate that placers exhibit consistent unnecessary suboptimality deviations as previous experiments. This also indicates that HPWL deviations are not an artifact of increasing netlist statistics - as measured by the ratio of the number of nets to number of cells, or total pin cardinality - beyond typical values[5], but rather inherent in the suboptimal performance of the placers.

Suboptimality evaluation of RMST and RSMT Metrics

We also evaluate the performance of placers with respect to RMST cost, which serves as a heuristic approximation to RSMT. Our experimental testbed is set up as described in Subsection II.C.6. The hyperedge cardinality increase procedure is applied to increase the total cardinality by 20%, and then the RMST value is calculated with respect to the given original placement. Our results are reported in Table II.9. From the results, it is clear that the placers are also suboptimal when it comes to the RMST metric, in addition to HPWL as shown earlier.

II.E Conclusions

In this chapter, we have introduced a new concept, *zero-change netlist transformations*, to (1) calculate the exact suboptimality of existing placers on

[5]The reader should notice that the synthesized netlist has larger values of parameters than the netlist used to derive it, yet its statistics are identical, i.e., typical, with respect to the original unprocessed netlist.

artificial constructed instances, and (2) calculate the partial suboptimality of placers on netlists synthesized from arbitrary netlists. While one may envision many transformations that do not change the HPWL of a given netlist, our transformations share an important property: the optimal HPWL of the new netlist is not less than the original HPWL optimal value, and consequently the placement of the new benchmarks is not "easier" than for the original benchmarks. By applying our transformations and re-executing the placement task, we can interpret any deviation in HPWL results as a lower bound on the deviation from optimality of the HPWL value. Our set of netlist transformations can be summarized as follows.

- Zero-change HYPEREDGE CARDINALITY INCREASE increases, if possible, the cardinality of hyperedges with degree ≥ 3 while leaving 2-pin edges intact.
- Zero-change HYPEREDGE DECOMPOSITION simplifies large hyperedges (when possible) of degree ≥ 3 by decomposing a larger hyperedge into two or more smaller hyperedges.
- Zero-change EDGE SUBSTITUTION increases the number of two-pin edges if possible.

Our empirical results show that even when testing only a few variant netlists, we can easily find consistent deviations in placed wirelength, proving the suboptimal behavior of current placers. From our empirical results, we make the following general remarks.

- Remark 1: Increasing the cardinality of hyperedges leads to consistent suboptimal behavior from the placers.
- Remark 2: Empirical results indicate that placers exhibit highest amount of sensitivity with respect to edge substitution.
- Remark 3: The hyperedge decomposition transformation simultaneously increases the number of hyperedges while reducing the total pin count (or hyperedge cardinality). Thus, it has the potential to reduce the HPWL (from

Remark 1), and to increase the HPWL (from Remark 2). Our experimental results show negligible change in wirelength, thus we can surmise that possible improvement in the performance of the placer due to hyperedge cardinality reduction is counteracted by the increase in number of hyperedges.

- Remark 4: The suboptimal behavior of placers in response to zero-change transformations is not an artifact of the increase in netlist statistics of zero-change netlist transformations. Embedding zero-change netlist transformations in a flow that preserves netlist statistics clearly shows that placers exhibit the same suboptimal behavior even though basic netlist statistics are kept intact.

- Remark 5: Suboptimality trends that are demonstrated for HPWL are also demonstrated for RMST.

Experimental researchers in physical design would no doubt agree there is a tendency to tune algorithms and codes to specific benchmarks [4]. A good placer should be good not just for a single real instance, but also for "similar" instances. Using our transformations allows the creation of a range of instances around any given arbitrary benchmark. Thus, for the first time, the field is afforded a means of creating "similar" instances in a systematic way such that the consistency of placement quality can be immediately evaluated.

The material in this chapter is based on the following publication. Authors' names are listed in alphabetical order.

- A. Kahng and S. Reda, "Zero-Change Netlist Transformations: A New Technique for Placement Benchmarking," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*

Table II.4: Deviations in HPWL in response to 20% zero-change total hyperedge cardinality increase. Deviations are calculated with respect to each placer's original placement run.

bench-mark	Placer			
	Capo	FS	Dragon	mPL
ibm01	9.72%	5.55%	20.43%	7.28%
ibm02	10.55%	6.79%	-	10.31%
ibm03	13.17%	3.63%	6.67%	5.38%
ibm04	12.53%	3.86%	16.26%	8.72%
ibm05	3.75%	1.50%	2.84%	4.01%
ibm06	4.48%	3.66%	15.71%	13.74%
ibm07	8.30%	3.61%	16.85%	5.50%
ibm08	5.48%	9.34%	7.97%	9.48%
ibm09	10.52%	8.93%	24.51%	7.54%
ibm10	11.02%	3.43%	12.90%	11.22%
ibm11	22.49%	5.40%	17.98%	24.42%
ibm12	4.14%	3.71%	11.14%	22.49%
ibm13	6.02%	4.33%	7.43%	6.10%
ibm14	16.73%	3.18%	15.80%	12.34%
ibm15	9.97%	3.03%	9.04%	12.71%
ibm16	5.30%	7.89%	29.11%	18.38%
ibm17	11.69%	4.87%	17.09%	14.18%
ibm18	7.22%	3.54%	20.29%	16.10%
Average	9.62%	4.79%	14.82%	11.66%

Table II.5: HPWL deviations in response to zero-change hyperedge decomposition. Deviations are calculated with respect to each placer's original placement run.

bench-mark	Placer							
	$\frac{w_2-w_1}{w_1}$ results				$\frac{w_3-w_1}{w_1}$ results			
	Capo	FS	Dragon	mPL	Capo	FS	Dragon	mPL
ibm01	4.69%	-	3.52%	0.17%	1.99%	-	2.04%	-1.33%
ibm02	2.08%	6.75%	18.83%	3.30%	0.32%	4.62%	12.68%	2.31%
ibm03	2.19%	1.83%	2.12%	0.30%	0.98%	0.98%	1.05%	-0.62%
ibm04	2.17%	3.54%	-0.20%	-3.15%	0.84%	2.37%	-0.97%	-4.18%
ibm05	7.19%	-0.28%	0.72%	-0.74%	5.61%	-0.82%	0.14%	-1.21%
ibm06	2.61%	1.62%	4.06%	-0.61%	0.88%	0.70%	2.88%	-2.05%
ibm07	2.93%	2.77%	0.06%	-0.73%	1.48%	1.84%	-1.08%	-1.77%
ibm08	2.64%	-0.77%	3.25%	3.75%	1.42%	-1.55%	2.26%	2.71%
ibm09	2.78%	2.22%	2.97%	0.84%	0.83%	0.96%	1.79%	-0.31%
ibm10	8.64%	3.37%	6.90%	9.45%	6.63%	2.02%	5.48%	7.95%
ibm11	2.61%	-0.78%	2.95%	0.37%	1.05%	-1.69%	1.98%	-0.82%
ibm12	3.49%	2.90%	0.73%	0.84%	1.87%	1.64%	-0.43%	-0.36%
ibm13	3.64%	2.15%	3.09%	1.08%	2.10%	1.02%	2.12%	0.05%
ibm14	2.94%	0.94%	2.25%	-1.23%	1.60%	0.02%	1.11%	-1.23%
ibm15	3.97%	1.96%	0.34%	0.97%	2.48%	0.85%	-0.74%	-0.07%
ibm16	2.64%	2.63%	-0.09%	11.62%	1.00%	1.38%	-1.34%	10.13%
ibm17	4.03%	1.77%	3.47%	2.74%	2.65%	0.81%	2.32%	1.59%
ibm18	1.70%	2.53%	7.67%	1.13%	0.23%	1.59%	6.62%	0.09%
Average	3.50%	2.27%	3.48%	1.67%	1.89%	0.98%	2.11%	0.60%

Table II.6: HPWL deviations in response to zero-change edge substitution. Number of nets increases by 10% for all benchmarks. Deviations are calculated with respect to each placer's original placement run.

bench-mark	Placer			
	Capo	FS	Dragon	mPL
ibm01	5.24%	6.09%	11.85%	5.76%
ibm02	5.99%	8.35%	-	8.36%
ibm03	8.66%	3.44%	4.67%	1.64%
ibm04	6.47%	4.83%	3.72%	10.61%
ibm05	2.38%	1.13%	1.99%	1.60%
ibm06	4.59%	2.62%	6.65%	4.90%
ibm07	5.50%	4.14%	7.68%	3.46%
ibm08	5.57%	2.13%	7.18%	4.54%
ibm09	9.48%	7.05%	12.64%	5.12%
ibm10	7.78%	3.80%	9.53%	3.90%
ibm11	10.34%	5.93%	16.57%	16.20%
ibm12	5.38%	4.00%	8.62%	9.16%
ibm13	9.86%	5.81%	7.91%	14.50%
ibm14	14.27%	12.03%	13.54%	12.94%
ibm15	3.49%	2.83%	8.97%	11.27%
ibm16	8.50%	5.27%	9.08%	9.54%
ibm17	6.63%	5.24%	11.02%	5.94%
ibm18	9.64%	3.40%	10.74%	6.60%
Average	7.21%	4.89%	8.96%	7.56%

Table II.7: HPWL deviations in response to hybrid zero-change netlist transformations. Deviations are calculated with respect to each placer's original placement run.

bench-mark	Placer			
	Capo	FS	Dragon	mPL
ibm01	8.52%	2.31%	9.05%	4.70%
ibm02	6.39%	-	-	8.12%
ibm03	11.99%	2.22%	3.26%	2.06%
ibm04	8.52%	1.59%	3.76%	11.31%
ibm05	1.67%	1.10%	1.08%	-0.31%
ibm06	5.43%	16.78%	5.02%	9.31%
ibm07	4.83%	2.83%	3.54%	2.81%
ibm08	13.78%	3.13%	9.93%	12.16%
ibm09	7.17%	7.24%	14.28%	3.56%
ibm10	10.95%	4.11%	8.92%	1.94%
ibm11	9.96%	2.04%	10.35%	11.10%
ibm12	3.55%	3.26%	10.62%	7.74%
ibm13	3.96%	6.60%	10.76%	10.02%
ibm14	11.19%	6.64%	16.35%	-
ibm15	8.25%	2.17%	7.76%	12.41%
ibm16	5.48%	3.72%	9.85%	7.27%
ibm17	9.70%	1.52%	7.77%	6.35%
ibm18	7.05%	2.17%	9.19%	12.37%
Average	7.07%	4.08%	8.32%	7.23%

Table II.8: HPWL deviations in response to netlist-statistics preserving zero-change netlist transformations. Transformed netlists have the same total number of nets and total hyperedge cardinality as the original netlists.

bench-mark	Placer			
	Capo	FS	Dragon	mPL
ibm01	6.74%	3.98%	10.82%	0.00%
ibm02	4.90%	18.96%	0.00%	0.30%
ibm03	6.29%	5.29%	8.25%	11.17%
ibm04	5.08%	6.40%	4.00%	-
ibm05	0.91%	0.69%	0.64%	1.50%
ibm06	2.26%	4.63%	5.95%	4.20%
ibm07	4.05%	5.79%	9.13%	2.84%
ibm08	3.06%	6.52%	8.77%	6.74%
ibm09	4.95%	5.23%	7.04%	14.92%
ibm10	5.08%	4.90%	9.04%	6.82%
ibm11	6.45%	4.59%	7.97%	3.89%
ibm12	5.78%	4.69%	5.53%	6.39%
ibm13	4.64%	3.56%	9.79%	5.76%
ibm14	9.48%	11.02%	14.97%	5.98%
ibm15	7.33%	2.41%	9.98%	13.34%
ibm16	8.10%	8.46%	10.57%	14.57%
ibm17	5.09%	7.12%	7.17%	12.87%
ibm18	5.31%	3.39%	12.24%	11.94%
Average	5.31%	5.98%	7.88%	7.24%

Table II.9: RMST deviations in response to hyperedge cardinality transformations. Deviations are calculated with respect to pre-calculated RMST values from the placer's original placement run.

bench-mark	Placer			
	Capo	FS	Dragon	mPL
ibm01	12.20%	4.07%	15.87%	4.62%
ibm02	9.01%	2.77%	43.46%	6.42%
ibm03	15.00%	2.74%	7.75%	9.14%
ibm04	9.97%	2.36%	10.09%	5.79%
ibm05	2.82%	1.56%	2.18%	3.38%
ibm06	4.86%	2.11%	12.03%	5.83%
ibm07	5.53%	3.37%	13.39%	4.70%
ibm08	7.48%	4.79%	13.98%	4.52%
ibm09	8.76%	13.17%	7.34%	5.41%
ibm10	7.96%	3.15%	13.22%	13.93%
ibm11	7.41%	3.05%	15.54%	9.32%
ibm12	15.88%	2.47%	7.46%	8.06%
ibm13	7.14%	5.24%	16.60%	10.08%
ibm15	10.51%	2.21%	15.86%	18.96%
ibm16	11.48%	6.11%	17.59%	21.86%
ibm17	12.08%	13.35%	7.71%	15.14%
ibm18	6.87%	3.29%	17.91%	16.64%
Average	8.61%	4.21%	13.22%	9.10%

Chapter III

CMOS Placement Improvement Via Feedback

As proved in the previous chapter, current placer solution quality is not close to optimal. This apparent performance gap needs to be addressed. In this chapter, we focus on improving top-down min-cut placers which have become a favored choice for modern placer implementations [16, 92, 98]. This choice is mainly motivated by the availability of strong multi-level partitioners, as well as the excellent scalability and runtime promise of the top-down paradigm. Despite the suboptimal behavior of top-down min-cut placers, the multi-level partitioners which are the main engines for top-down min-cut placers are close to optimal [23, 22]. This raises the question of how excellent partitioner performance may be "mistranslated" into far from excellent placement performance.

To understand this question one must examine the main components – apart from multi-level partitioners – that determine a min-cut placement result. These components include (i) top-down paradigm (ii) cut-sequence, and (iii) terminal propagation. If we examine the first component, i.e., the top-down paradigm, then an obvious question is whether a 2^k-way partitioner gives far better results than executing a 2-way partitioner for k levels. This has been answered in the negative by Karypis and Kumar [60]. The second component, cut-sequences, have enjoyed much recent attention [16, 99, 5]. Caldwell et al. [16] suggest using bin

aspect ratio as the decisive factor in determining cut direction; this leads to flexible slicing floorplan structures rather than the traditional horizontal-vertical alternation. Bin aspect ratio has also been explored via a dynamic-programming based approach [99] and fractional cut sequences [5], which lead to further reductions in wirelength. The third item, terminal propagation, has not enjoyed much investigation, yet it is decisive, since it is responsible for translating the partitioner results into global placement wirelength assumptions. Few works address terminal propagation [30, 85, 41, 16, 17, 92], and mostly follow the initial approach of Dunlop and Kernighan [30]. Other approaches try to omit terminal propagation altogether and opt for global or exact wirelength objectives [41, 100, 98]. Accurate terminal propagation is the subject of this chapter.

We define *ambiguous* terminal propagations as propagations arising from terminals that lie equally proximate from two sub-bins of a bin being partitioned, so that their destination propagation is ambiguous. To reduce this ambiguity, we carefully re-examine the repartitioning problem [41] and show that it is abstractly a form of *placement feedback*, where future cell locations are used to determine present terminal propagation results. Since these terminal propagations produce new results that change the output results, the feedback can be iterated a number of times in order to attain stable and consistent improvements. We propose and investigate variant "feedback controllers" to fine-tune the placement response and optimize wirelength. We summarize our contributions as follows.

- We re-examine the repartitioning problem [41] (without overlapping) in the context of top-down recursive-bisection placement and quantify its effect on the number of ambiguous propagations.
- We show that the problem is similar to feedback systems.
- We propose to iterate the number of repartitions according to a number of different objectives, i.e., controllers.
- We develop efficient implementations [52, 54] in the open-source Capo placer,

and empirically demonstrate the improvement in Capo's performance using both regular circuits and our ZCNT benchmarking framework.

The organization of this chapter is as follows. In Section III.A we examine the top-down min-cut placement methodology and its essential component of terminal propagation. In Section III.B we present our feedback methodology for accurate terminal propagation control. Section III.C gives experimental results on various standard benchmarks. Finally, Section III.D summarizes the merits of our feedback approach.

III.A Background

In this section we give a brief overview of top-down min-cut placement as well as the necessary background for terminal propagation.

III.A.1 Top-Down Min-Cut Placement

In min-cut placement, a placement region is a collection of *bins*. Each bin corresponds to a fixed rectangle into which nodes of a hypergraph should be placed. Initially, the chip's core region is comprised of one bin. The min-cut placement methodology proceeds by recursively partitioning each bin and its associated hypergraph, and assigning the partitioned subhypergraphs to sub-bins. All nodes (or cells) that are assigned to a sub-bin are considered, for wirelength estimation and terminal propagation purposes, to be placed at the geometric center of the bin. Partitioning usually alternates between vertical and horizontal cuts, or as determined by the bin aspect ratio [87, 16, 99]. The product of the partitioning process is a slicing floorplan as shown in Figure III.1. The partitioning process continues until a certain bin threshold size, beyond which end-case placers [15] are used to assign actual locations of hypergraph nodes in their corresponding bins. Given a set of disjoint bins whose union is the entire placement region, we use the term *placement level* partitioning to indicate the process of partitioning each bin

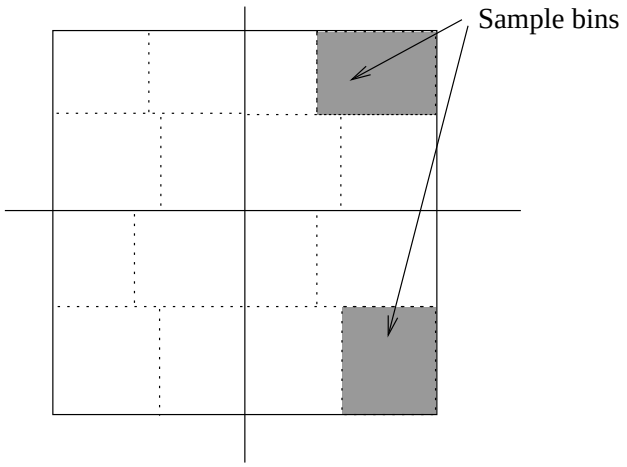


Figure III.1: A snapshot of a min-cut placement. Solid horizontal lines represent first-level cuts, and solid vertical lines represent second-level cuts. Dashed horizontal lines represent third-level cuts, and dashed vertical lines represent fourth-level cuts.

exactly once. Hence, the whole min-cut top-down placement methodology can be considered as the progression of placement levels from a coarse top level down to a fine bottom level.

III.A.2 Terminal Propagation

Terminal propagation [30] is the process through which nodes external to a given bin being partitioned are propagated as fixed terminals (nodes) to that bin. These terminals bias the partitioner toward placing movable nodes close to their terminals, thus reducing placement wirelength. Given a bin being partitioned to two sub-bins and a node externally connected to this bin, the sub-bin to which this node is propagated as a terminal is typically determined by (1) calculating the distances between the node's position and the centers of the two new sub-bins, and (2) with some tolerance, propagating the node to the closer center as a fixed terminal.

Consider some bin B being partitioned into two sub-bins B_1 and B_2, with the geometrical center of each bin denoted by $c(B_1)$ and $c(B_2)$. Given a net e with

some set of cells e_i in B and some set of cells e_x in other bins, we can partition e_x into three sets:

1. $e_x^1 \subseteq e_x$ is the set of cells that are geometrically closer to $c(B_1)$ than $c(B_2)$. We measure the distance in Manhattan norm.
2. $e_x^2 \subseteq e_x$ is the set of cells that are geometrically closer to $c(B_2)$ than $c(B_1)$.
3. $e_x^3 \subseteq e_x$ is the set of cells that are equally proximate to $c(B_2)$ and $c(B_1)$. This is computed with some tolerance, i.e., we consider two distances equal as long as the ratio between the two distances does not exceed a certain threshold δ_{fuzzy}.

Given the previous definitions, terminal propagation decisions can be summarized as follows:

- **Case (1):** If $e_x^1 \neq \emptyset$ and $e_x^2 = \emptyset$ then a fixed cell of zero weight is added to the center of B_1 and connected to e_i via a hyperedge.
- **Case (2):** If $e_x^1 = \emptyset$ and $e_x^2 \neq \emptyset$ then a fixed cell of zero weight is added to the center of B_2 and connected to e_i via a hyperedge.
- **Case (3):** If $e_x^1 \neq \emptyset$ and $e_x^2 \neq \emptyset$ then no terminals are propagated.
- **Case (4):** If $e_x^1 = \emptyset$, $e_x^2 = \emptyset$, and $e_x^3 \neq \emptyset$ then no terminals are propagated in this case [30, 3], or one fixed cell is added to B_1, another fixed cell is added to B_2, and both are connected to e_i [17, 16]. We call this case *ambiguous terminal propagation*.

The following example illustrates terminal propagation decisions.

Example III.1: If a bin B is being partitioned into sub-bins B_1 and B_2 as shown in Figure III.2, then any nodes in bin C that are connected to nodes in B will be propagated as fixed nodes to B_1 as shown. There is no ambiguity about this

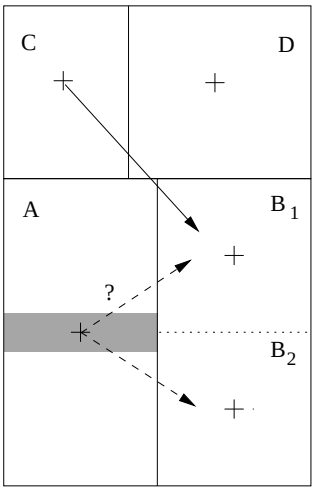


Figure III.2: Example of terminal propagation.

propagation, and terminal propagations from any future bisections within bin C will continue to be propagated to bin B_1. However, for some nodes this cannot be decided accurately. For example, all nodes in bin A are equally proximate to both sub-bin centers of bin B. These nodes lead to ambiguous terminal propagation. As indicated earlier, the traditional solution is to propagate such nodes to both su-bbin centers [17, 16], or not to propagate at all [30, 3]. The intuition behind these propagation approaches is that it is better to make no decision rather than a bad decision.

Ambiguous terminal propagations can lead to partitioning results that do not capture the global objective of wirelength minimization. If $e_x^3 \neq \emptyset$ then the terminal propagation decision becomes inaccurate: Case (1) can likely be Case (3), Case (2) can likely be Case (3), and Case (4) can be any of Cases (1), (2), or (3). In general, the proximity of a node to a sub-bin center is calculated with some tolerance δ_{fuzzy} (recently referred to as *partition fuzziness* in [3]). In Capo [16], this partition fuzziness was originally set to 10%, then later revised to 33% [3]. This latter tolerance matches the value suggested by [30]. The increased fuzziness essentially increases the number of ambiguous terminal propagations to

avoid making bad decisions.

To eliminate the dependency of the placement problem on terminal propagation, Huang and Kahng [41] introduced exact objectives (e.g., minimum spanning tree models for net routing) to drive the partitioning process. In particular, *net vectors* are used as a means to quantify the global contribution of each cut and to eliminate the need for propagation. Another idea introduced by [41] is *cycling* of the partitioning process, by forming a *sliding window* which goes over the bins and repartitioning them since the results of partitioning introduce new terminal locations and hence different minimum spanning tree costs. The sliding window also overlaps in its movement allowing cells to migrate from their assigned bins. Also, Zhong and Dutt [100] and Yildiz and Madden [98] used global half-perimeter wirelength objectives to drive the partitioner. Zhong and Dutt give experimental results showing improvements versus terminal propagation-based approaches, at the expense of increased runtime; Yildiz and Madden conclude that wirelength improvements using their approach are modest.

A top-down placement flow using terminal propagation can be conceptually represented as in Figure III.3(a). The input to the placement is the set of nodes initially placed at the center of the core placement region. Each placement level is divided into two steps: terminal propagation and bin partitioning.

III.B Accurate Terminal Propagation

III.B.1 The Ambiguous Terminal Propagation Problem

The purpose of this work is to mitigate the effects of ambiguous terminal propagations, i.e., we would like to eliminate or minimize propagations corresponding to Case (4) when making terminal propagation decisions. While it may seem that the contribution of these propagations is small in the overall top-down min-cut placement approach, our analyses and results indicate that these propagations can have tremendous impact on the final wirelength and quality of the min-cut placement. Also, while it might seem possible that reordering bin processing can

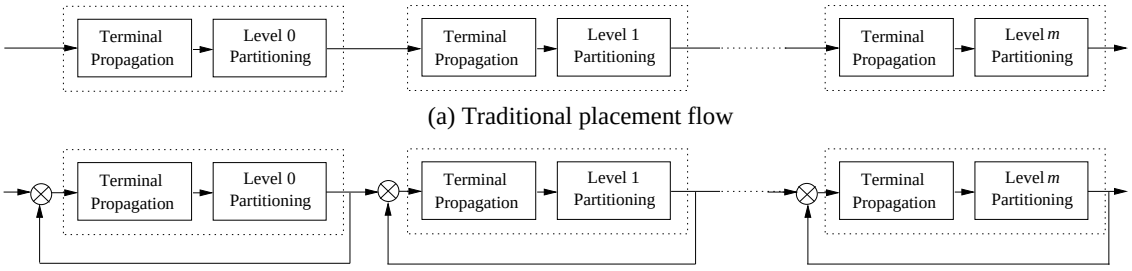


Figure III.3: A view of the top-down placement process.

reduce the total amount of ambiguous terminal propagations, our experimental results indicate otherwise. We will later examine the issue of bin ordering in Subsection III.B.5. We now propose how to mitigate the effects of ambiguous terminal propagation using the concept of *placement feedback*.

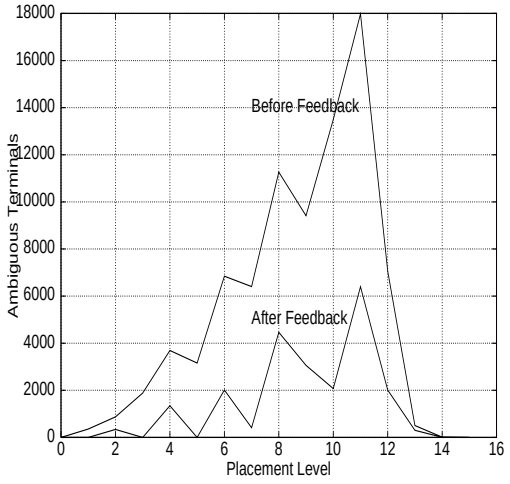
III.B.2 Placement Feedback

We define *placement undoing* as merging two sub-bins that were originally partitioned, so that they are one bin again. Placement undoing enables us to realize accurate terminal propagation. At each level of placement, all bins are partitioned. After such partitioning, we undo all the partitioned bins but we keep the node locations as assigned by the partitioning. That is, we decouple (1) the placement of a node for use in terminal propagation from (2) its bin location. We then use the new accurate node locations to re-do bin partitioning and update the node locations as necessary, i.e., the output of the placement level is taken back as its input. This can be conceptually regarded as a *feedback loop* within each placement level, as shown in Figure III.3(b): this feedback takes the current result of a placement level and feeds it back to the input while undoing the placement. This resembles the flow in Figure 1 of [5]. The following example provides an illustration.

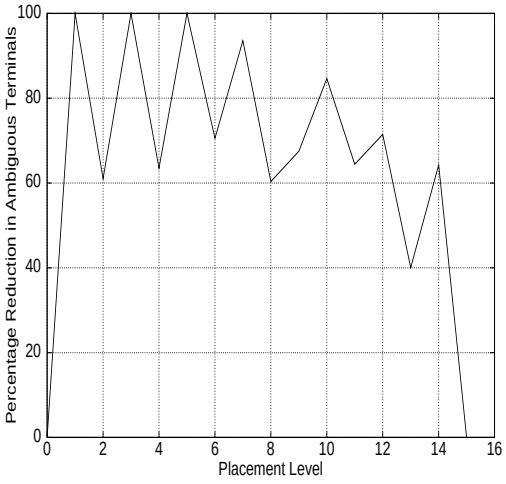
Example III.2: If bin B is being partitioned into two sub-bins B_1 and B_2 as shown in Figure III.2, then we partition bin B, propagating nodes in bin A to

both B_1 and B_2 (ambiguous propagation). We then partition bin A (into two sub-bins A_1 and A_2), as well as bins C and D. Now that the whole placement level is partitioned, we undo all bin partitionings, restoring the original structure. Despite our having undone the partitioning, we keep the node locations as given by the partitioning results. We use these new locations as input to re-do the partitioning, where in this case no ambiguous terminal propagation occurs. The final node locations are adjusted according to the re-done partitioning results. We stress that feedback only alters the terminal propagation results of *ambiguous* propagations. For example, the propagation locations of nodes propagated from bin C to bin B will not change when we apply feedback. It is only ambiguous propagations from bin A to bin B that benefit from such feedback.

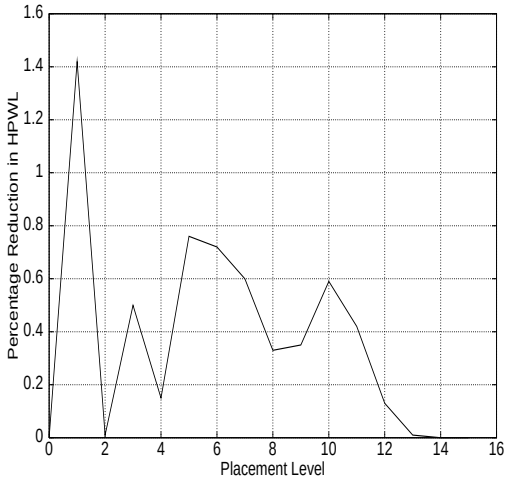
We empirically examine the relation between reductions in ambiguous terminal propagations and wirelength reduction as measured by HPWL. We implement placement feedback in a well-established top-down min-cut placer, Capo (Version 8.7 [16, 1]). Our changes take 130 lines of code. We report two metrics: (i) the percentage reduction in ambiguous terms per placement level, i.e., we calculate the number of ambiguous terminals before and after feedback, and (ii) the percentage reduction of HPWL per placement level, i.e., we calculate the percentage reduction in the HPWL estimate of each level (assuming, as is standard, pin locations at bin centers). For the ibm01 benchmark [1], we report the actual number of ambiguous terminal propagations before and after feedback in Figure III.4(a), the percentage reduction in ambiguous terminals in Figure III.4(b), and the percentage reduction in half-perimeter wirelength (HPWL) in Figure III.4(c). The two percentage reductions in Figures III.4(b) and III.4(c) are well-correlated with each other, lending support to our intuition. The total number of ambiguous propagations across all placement levels drops from 82947 terminals to 22435 terminals, a reduction of around 73%. In another experiment, we quantify the contribution of each level's feedback loop to the final HPWL. To do this, given a level i, we enable the feedback loops for all levels up to level i and disable all remaining $m - i$ loops, where m is the total number of placement levels, and cal-



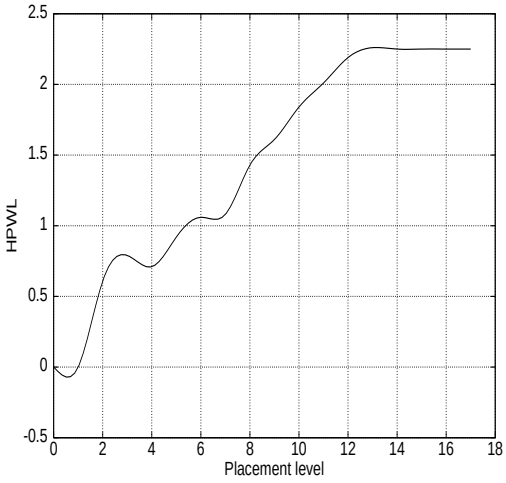
(a) Number of ambiguous terminals before and after feedback for each placement level.



(b) Percentage reduction in ambiguous terminals for each placement level.



(c) Percentage reduction in wirelength for each placement level.



(d) Cumulative percentage reduction in wirelength after each placement level.

Figure III.4: Relation between wirelength and ambiguous terminal reduction and placement level.

culate the final HPWL. Our results are given in Figure III.4(d) for all levels of the ibm01 benchmark. These results are averages of 6 runs with different random seeds. From this figure, we observe that except for the last, very few placement

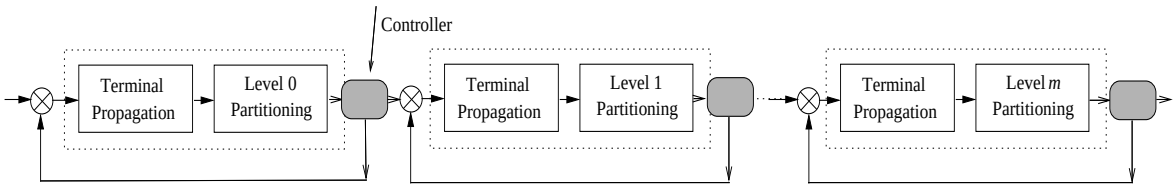


Figure III.5: A feedback system with controllers.

levels, reductions in HPWL increase almost linearly with each placement level. We next examine how to fine-tune the placement feedback via the concept of *feedback controllers*.

III.B.3 Iterative Controlled Placement Feedback

Since the feedback loop produces new outputs, it is natural to iterate over the feedback loop a number of times until one attains the most accurate terminal propagation and hence the best overall reduction in HPWL. The problem of feedback systems is that the output might not be predictable, i.e., the system can loop infinitely or in the best case converge rapidly to the final stable output [29]. Typically, if the feedback response is not desirable then some *feedback controller* is inserted to enhance the response, as shown in Figure III.5. We propose a number of controllers that are suited to the placement problem.

For our purposes, a feedback controller function controls the response of a placement level by measuring some quality Q at the output of the placement level, and feeding back corrective information to the input of the placement level so as to optimize the quality Q. The controller might also decide to stop sending feedback information, i.e., terminate the looping, if it no longer perceives improvement in the measured quality Q. We propose and motivate two quality measures that can be chosen as objectives during feedback.

1. **HPWL quality Q_H:** Q_H is the value of the HPWL estimate at a particular placement level. Since the general placement objective is to minimize wirelength

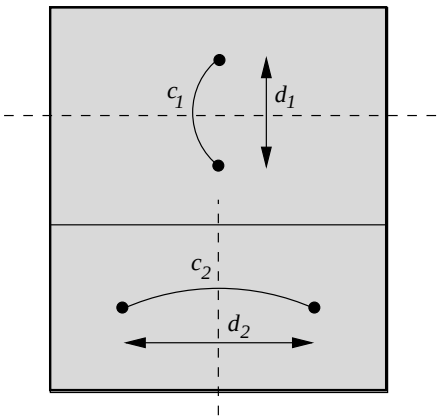


Figure III.6: The discrepancy between the partitioning quality and the HPWL estimate. Partitioning quality is equal to $c_1 + c_2$, while HPWL estimate is equal to $c_1 d_1 + c_2 d_2$.

or HPWL, a feedback controller might seek to optimize a placement level based on the HPWL estimate.

2. **Partitioning quality Q_P:** Q_P is the sum of all partition cuts at a particular placement level. The motivation of this objective is that during the early placement levels, the HPWL estimate (based on bin center locations) can be very inaccurate; partitioning results might be more relevant since initial good partitions are likely to lead to good final HPWL results.

While it may intuitively seem that optimizing Q_P or Q_H entails optimizing the other, this might not be the case. The following example illustrates the subtle difference between these two qualities.

Example III.3: In Figure III.6, we have two bins being partitioned as indicated by the dashed lines. These two bins are the outcome of an initial partitioning indicated by the horizontal solid line. Assume that after the first partitioning we obtain cut values c_1 and c_2 as shown in the figure. If the distance between the centers of the child bins of the top and bottom bins are d_1 and d_2 respectively, then $Q_P = c_1 + c_2$ and $Q_H = c_1 d_1 + c_2 d_2$, assuming no connection between the upper

and lower bins. After feedback, these cut values change to c_1' and c_2', and hence $Q_P' = c_1' + c_2'$, while $Q_H = c_1'd_1 + c_2'd_2$. If $c_1' < c_1$, but $c_2' > c_2$, then the change in Q_H depends on the values of d_1 and d_2. For example, if $c_1 = 100,\ c_2 = 100,\ d_1 = 6$, and $d_2 = 8$ then $Q_P = 200$ and $Q_H = 1400$, and after feedback $c_1' = 85,\ c_2' = 112$ then $Q_P' = 197$ and $Q_H' = 1406$. Thus, the partitioning quality improves but the HPWL estimate does not improve. A feedback controller that optimizes Q_H will definitely choose the first partitioning as the best feedback loop, while a controller that optimizes Q_P will choose the second partitioning as the best feedback loop.

In addition to the placement objective to be optimized, a controller must decide when to stop iterating and feed results from its placement level forward to the next placement level. We propose and empirically evaluate three kinds of controllers. We assume that each feedback loop is executed at most k times.

1. **Monotonic Improvement Controller:** In this scheme, the controller fixes some quality to improve, either Q_P or Q_H, and keeps on iterating over the feedback loop until there is no further improvement in the quality measure, i.e., the controller loops as long as Q_H or Q_P continues to decrease. The controller stops iterating if an increase in Q_P or Q_H is observed, and then passes the previous partitioning results to the next placement level.

2. **Best Improvement Controller:** In this scheme, the controller fixes some quality to improve, either Q_P or Q_H, and allows k iterations over the feedback loop. After finishing k loops, the controller passes to the next placement level the results of the *best* iteration seen (in terms of the chosen quality measure). Notice that normally, the controller does not feed back its best results; it always feeds the current output back to the input. However, the best improvement controller passes the best results seen in k feedback iterations to the next placement level.

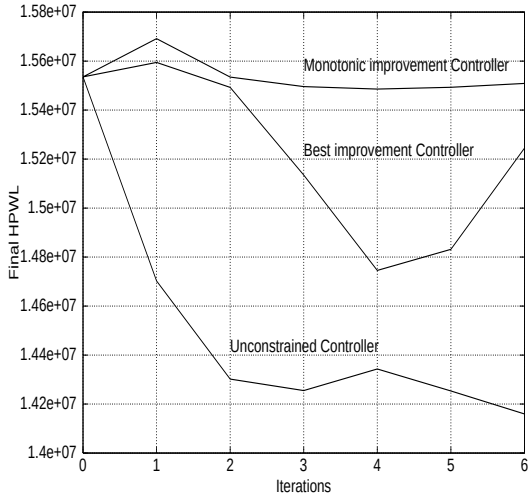
3. **Unconstrained Controller:** In this scheme, the controller allows feedback to follow its natural course over the k iterations and then passes the result of the last iteration to the next placement level.

Table III.1: Iterative feedback example. Q_H indicates the placement level quality as measured by the HPWL estimate. Q_P indicates the placement level quality as measured by the sum of partitioning results.

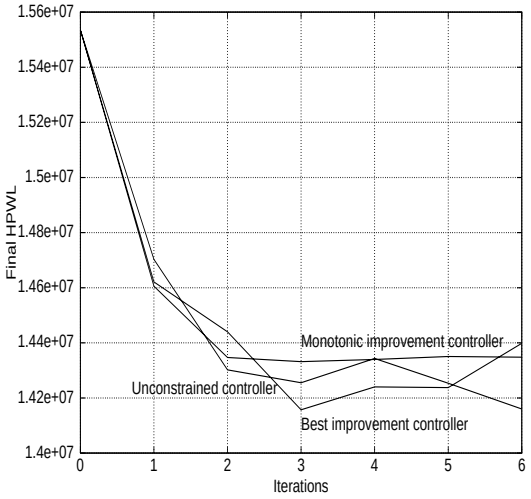
Level	Objective	Iteration					
		0	1	2	3	4	5
2	Q_H	527444	522734	527649	531745	527854	531955
	Q_P	3596	3071	3117	2567	2304	3128
3	Q_H	638241	855828	839084	834140	836531	837065
	Q_P	3991	2277	2188	2160	2189	2166
8	Q_H	1355790	1376390	1372710	1371780	1371990	1372760
	Q_P	6836	6054	5655	5614	5570	5485

We illustrate the behavior of iterative feedback and the operation of various controllers through the next example.

Example III.4: We fix the number of allowable iterations to $k = 5$, and observe a number of placement levels' outputs (as measured by Q_P and Q_H) for the ibm02 benchmark. We tabulate the results in Table III.1. The operation of the controllers is illustrated by using the Q_P objective. Iteration 0 indicates no feedback loop traversal, i.e., just the regular top-down partitioning. At placement Level 2, the monotonic improvement controller stops at Iteration 2, passing the placement result of $Q_P = 3071$; the best improvement controller stops after Iteration 5 but passes the best placement result seen ($Q_P = 2304$); and the unconstrained controller passes the last placement of $Q_P = 3128$. At placement Level 3, the monotonic improvement controller stops at Iteration 4, passing the placement result of $Q_P = 2160$; the best improvement controller stops after Iteration 5 and passes the best placement of $Q_P = 2160$; the unconstrained controller stops after six feedback loops and passes the placement with $Q_P = 2166$.



(a) Effect of iterative feedback with various controllers using the Q_H objective on the *final* HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final placement HPWL.



(b) Effect of iterative feedback with various controllers using the Q_P objective on the *final* HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final placement HPWL.

Figure III.7: Effect of controller choice on the final HPWL.

The last example shows the effect of iterative feedback and controller behavior on individual levels. Further studies examine the final HPWL after all placement levels, i.e., we measure the aggregate effect of all feedback loops and controllers on the final HPWL of the placement. We study the impact of both the allowable feedback iterations and the controller type on the quality of the final placement as measured by HPWL. Results for the same benchmark, ibm02, are plotted in Figure III.7. We plot results of controllers based on the HPWL objective, Q_H, in Figure III.7(a), and results of controllers based on the partitioning objective, Q_P, in Figure III.7(b). Notice that results of the unconstrained controller are identical in both Figure III.7(a) and Figure III.7(b). In Figure III.7, the horizontal axis represents the number of feedback iterations; Iteration 0 represents Capo's results with no feedback. All results represent an average of 4 seeds. From our experiments, we notice the following:

- Controllers based on the partitioning objective Q_P perform considerably better than controllers based on the HPWL objective Q_H. This supports the argument that initial good partitioning results likely lead to final good placements. The HPWL-based controllers are relatively inaccurate at early placement levels and are sensitive to the relative distances between bins. However, they yield better results in the late placement levels. Consequently, we have also tried *hybrid* approaches that apply the Q_P objective during the early placement levels and Q_H during the late placement levels, but the resulting improvement in quality of results was insignificant.

- The unconstrained controller performs as well as the Q_P best improvement controller since partitioning results tend to improve as the number of feedback iterations increase.

- The monotonic improvement controller usually takes less runtime since it can iterate for fewer than the k allowed feedback iterations.

- Overall, after a mere 3 iterations, the best improvement controller (or unconstrained controller) reduces Capo's final HPWL by about 8%. We have found that the controllers exhibit similar behavior on other benchmarks.

To expand the study of the relationship between final HPWL and number of iterations, we calculate the final HPWL for iterations up to 12, as an approximation to asymptotic analysis. We plot for the ibm02 benchmark the average results of 4 seeds in Figure III.8, smoothing the plot using cubic splines. From the plot, the aggregate final response of the unconstrained controller, as measured by the HPWL estimate, oscillates slightly around a fixed value. We conclude that iterated controlled feedback succeeds in eliminating the indeterminacy associated with ambiguous terminal propagations, and transforms the individual placement-level response into an overall stable mechanism that affords 8-9% HPWL improvement.

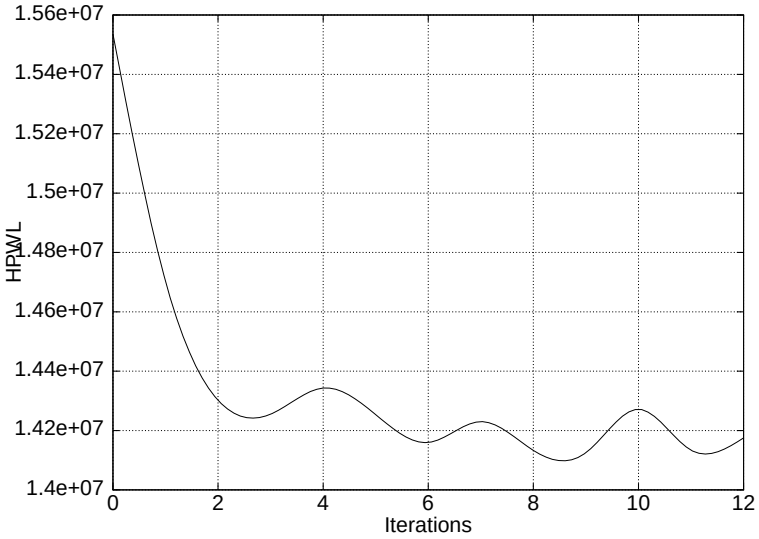


Figure III.8: Effect of iterative feedback using the *unconstrained controller* on the *final* HPWL of the ibm02 benchmark. The horizontal axis represents the number of allowable feedback iterations, while the vertical axis represents the final placement HPWL. The plot shows the average of results of 4 seeds. Up to 12 iterations are reported as an approximation to asymptotic analysis.

III.B.4 Accelerated Feedback

A drawback of placement feedback is the increased runtime. We propose a simple idea to reduce the runtime. Typically for each bin partitioning, placers execute calls to the multi-level partitioner a number of times and use the best reported results. For instance, Capo calls MLPart [8] twice to construct two cluster trees but only utilizes the best cluster-tree partitioning results. We notice that in iterated feedback, it is only the last feedback loop that actually determines the partitioning results; other loops determine accurate locations for ambiguous terminals. Hence, in order to speed up our feedback implementation, we call MLPart once for each feedback loop while restoring default Capo settings for the last feedback iteration. As the experimental results in Section III.C demonstrate, this improves runtime considerably.

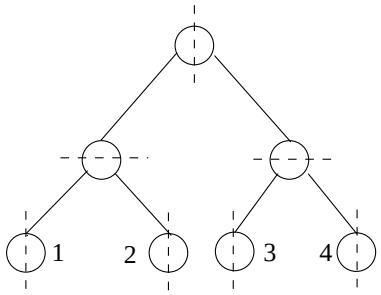


Figure III.9: Bin ordering strategies. Circles represent bins. Dashed lines represent the cut orientation.

III.B.5 Effect of Bin Ordering

We also consider the effect of bin ordering on the performance of placement feedback. Our results are inconclusive since they show no improvement in response to a number of various bin ordering strategies. We have tried the following bin ordering strategies.

1. **Normal traversal.** Process the bins in the natural order, i.e., as they appear in the top-down tree hierarchy. For example, in Figure III.9, the bin ordering would be 1, 2, 3, 4.

2. **Random traversal.** Process the placement bins in a random order. One random ordering in Figure III.9 is 1, 4, 3, 2. The intuition behind this approach is to break any adversarial order dependency to which a fixed ordering might be susceptible.

3. **Alternating traversal.** Process the ordering normally in one feedback iteration, and then reverse the direction of processing in the next iteration – hence the name, *alternate* processing. For example, in Figure III.9, one feedback iteration would process the bins in the order 1, 2, 3, 4, while the next iteration would process the bins in the reverse order, 4, 3, 2, 1. The intuition behind this approach is to consider all mutual dependencies, i.e, if some bin j's partitioning results depend on some bin i's partitioning results, then this dependency will be taken into account in one of the traversal directions.

We have implemented these ordering strategies, but obtained no better results than those attained by normal traversal.

III.C Experimental Results

Our heuristic is easy to implement and only linearly increases runtime by the number of feedback iterations executed. While there are a number of academic top-down min-cut placers such as Capo [16], Dragon [92], and Feng-Shui [98], we choose to implement our technique in Capo due to its code availability, excellent scalability, fast runtimes, and modular code design. We implement our technique in Capo, version 8.7. Our implementation requires 130 lines of C++ code. We report experimental results on four benchmark sets: IBM Version 1 (2% whitespace) [1], Peko [19] (Suite 1), and IBM Version 2 [97] (easy and hard instances). We run our experiments on a 2.4 GHz Xeon Linux workstation with 2 GB memory.

III.C.1 Evaluation Using Standard Benchmarks

In the first series of experiments we evaluate our technique on the IBM Version 1 benchmarks [1] (2% whitespace) and give the results in Tables III.2 and III.3. We report results of original Capo as indicated by **Mode Capo**, Capo with accelerated placement feedback as indicated by **Mode AFB**, and regular feedback as indicated by **Mode FB**. For all experiments, $k = 3$ iterations of unconstrained feedback are used. All runtimes are reported in seconds as indicated by the label **CPU (s)**. We give the best and average of 6 runs each with a different random seed, and report the percentage improvement in HPWL for both the best and average results[1]. From the table, the average improvements for accelerated feedback and regular feedback are 4.70% and 5.43% respectively. We also observe that HPWL improvements peak at nearly 14% for ibm05. Comparing runtimes, we find that accelerated feedback increases runtime to 2.02× Capo, and that feedback

[1]We have enabled rowIroning in both regular and feedback flows. RowIroning is a detailed placer within Capo that optimally re-places windows of 7-8 cells.

increases runtime to 3.13× Capo. We conclude that accelerated feedback significantly improves runtime with a small impact on solution quality as measured by HPWL.

In the second series of experiments we evaluate our technique on the PEKO benchmarks [19] and give the results in Tables III.4 and III.5. The column descriptions are identical to those of Tables III.2 and III.3. We notice that original Capo results as reported by [19] are different from the ones we report due to the release of a new version (version Capo 8.7 [3] rather than the old version 8.0 used by [19]). We can see that our technique successfully obtains further reductions in wirelength of up to 10%, with an average improvement of 5.37%, versus the latest Capo results.

Our third series of experiments evaluates the impact of our heuristic on both routability and final routed wirelength of the IBM version 2 [97] benchmarks, using Cadence's WRoute Version 2.4. We report the experimental results in Tables III.6 and III.7 for both IBM version 2 easy and hard instances. From the tables, our proposed heuristic improves the routability as measured by the number of violations for all instances. For example, WRoute smoothly routes the feedback placement of the ibm01 easy instance with 0 violations. Routability of ibm07 and ibm08 is also dramatically enhanced. These improvements in routability are likely due to the total reductions in wirelength. We also observe that routing of the feedback placements takes much less time than for Capo's original placements. Total placement and routing runtime for Capo is 50864 seconds, but this drops to 28901 seconds with feedback. Hence, the savings in routing time offset any runtime increase in placement due to feedback. As for wirelength, we can see that improvements reach up to 9% for ibm07h. The average improvement being obtained for routed wirelength of all benchmarks is 5.81% with the best results for the ibm01e and ibm07h testcases. These reductions in wirelength improve total congestion and power consumption. Comparing the number of vias, we find that feedback produces the least number of vias in most cases. The total number of vias for Capo is 3416×10^3, and 3362×10^3 vias with feedback. These reductions in number of vias may slightly improve both manufacturing yield and total delay.

III.C.2 Evaluation Using the ZCNT Benchmarking Approach

We evaluate the proposed feedback approach using the ZCNT framework proposed in the previous chapter. Using an IBM circuit, we first execute Capo with feedback to obtain the initial or reference placement. Then, the reference placement along with the circuit are fed to the ZCNT procedure (which changes the circuit statistics by 15% as described in Subsection II.C) to yield a new circuit. Capo is then executed twice on the new circuit – once with feedback and once without. The wirelength of the two new placements is then compared to the one obtained by the reference placement. We record all suboptimality deviations in Table III.8. From the table of results, Capo with feedback has an average deviation of about 5.27% (up to 15.03%), while Capo without feedback has an average deviation of about 9.43% (up to 23.75%). Thus the quality of placement has improved by 4.16% due to feedback. We conclude that placer improvement using feedback is manifested in both regular benchmarking circuits and in benchmarking using the ZCNT framework.

III.D Conclusions

In this chapter we have addressed the suboptimal placer behavior proved in the previous chapter. We have studied the problem of ambiguous terminal propagations, which introduces indeterminism in the placer performance. We have carefully re-examined the repartitioning problem [41] to diminish this indeterminism. We abstractly identified repartitioning as a form of *placement feedback*. In feedback, future node locations control present terminal propagation. This is realized by *undoing* a placement level after its partitioning, and feeding back the resultant node locations to the same placement level as input for partitioning. This feedback scheme is iterated, with a number of variant controllers to fine-tune the feedback response having been proposed and compared. Implementing our approach in Capo and applying it to standard benchmarks yields up to 14% HPWL

reductions (average 5.5%) for the IBM general (Version 1) benchmarks, up to 10% HPWL reductions (average 5.37%) for the PEKO (Suite 1) benchmarks, up to 9% routed wirelength reductions (average 5.8%) for the IBM (Version 2) benchmarks, and an average of 4.16% HPWL reductions in the ZCNT benchmarking framework. In addition, due to the reduction in wirelength, the proposed approach improves routability, routing runtime, and the number of vias.

The material in this chapter is based on the following publication. Authors' names are listed in alphabetical order.

- A. Kahng and S. Reda, "Wirelength Minimization for Min-Cut Placements via Placement Feedback," to appear in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Table III.2: Results for the IBM instances.

Circuit	Mode	CPU	Best	Impr	Average	Impr
ibm01	Capo	102	4.973		5.062	
	+ AFB	130	4.925	0.96%	4.990	1.43%
	+ FB	296	4.928	0.92%	4.967	1.88%
ibm02	Capo	189	15.39		15.53	
	+ AFB	259	14.11	8.31%	14.22	8.46%
	+ FB	588	13.94	9.41%	14.18	8.67%
ibm03	Capo	189	13.74		13.91	
	+ AFB	425	13.70	0.33%	13.80	0.74%
	+ FB	578	13.27	3.47%	13.65	1.82%
ibm04	Capo	238	17.96		18.17	
	+ AFB	529	16.99	5.41%	17.30	4.78%
	+ FB	699	17.07	4.97%	17.20	5.35%
ibm05	Capo	405	43.52		44.26	
	+ AFB	648	37.74	13.27%	38.31	13.45%
	+ FB	1214	37.63	13.52%	38.19	13.73%
ibm06	Capo	342	20.75		21.32	
	+ AFB	636	20.79	-0.17%	21.23	0.43%
	+ FB	1135	20.85	-0.49%	21.39	-0.32%
ibm07	Capo	483	35.25		35.94	
	+ AFB	702	32.19	8.68%	32.52	9.52%
	+ FB	1493	32.39	8.12%	32.64	9.18%
ibm08	Capo	510	38.03		38.21	
	+ AFB	792	34.51	9.25%	34.99	8.43%
	+ FB	1627	34.77	8.57%	35.44	7.24%
ibm09	Capo	543	29.69		30.53	
	+ AFB	730	28.23	4.93%	28.74	5.84%
	+ FB	1624	28.39	4.37%	28.66	6.11%

Table III.3: Results for the IBM instances.

Circuit	Mode	CPU	Best	Impr	Average	Impr
ibm10	Capo	740	59.89		61.48	
	+ AFB	1125	58.18	2.85%	58.92	4.15%
	+ FB	2309	58.06	3.05%	58.64	4.61%
ibm11	Capo	699	45.69		46.44	
	+ AFB	1683	44.46	2.69%	44.93	3.25%
	+ FB	2173	43.30	5.23%	43.88	5.51%
ibm12	Capo	1011	82.70		83.77	
	+ AFB	2063	75.62	8.56%	77.34	7.67%
	+ FB	3133	76.36	7.67%	77.08	7.99%
ibm13	Capo	966	54.90		55.96	
	+ AFB	2072	53.64	2.29%	54.54	2.54%
	+ FB	3260	54.19	1.29%	54.76	2.15%
ibm14	Capo	1642	129.6		131.8	
	+ AFB	4155	122.6	5.44%	124.6	5.45%
	+ FB	6034	122.1	5.80%	122.9	6.79%
ibm15	Capo	1708	143.4		145.1	
	+ AFB	5568	137.2	4.33%	138.8	4.39%
	+ FB	5610	137.2	4.32%	138.1	4.82%
ibm16	Capo	2050	185.3		187.0	
	+ AFB	6455	173.3	5.33%	175.8	5.65%
	+ FB	6735	173.7	6.31%	175.1	6.37%
ibm17	Capo	2791	275.5		281.5	
	+ AFB	7207	269.9	0.95%	272.7	2.87%
	+ FB	8078	267.2	3.01%	269.5	4.25%
ibm18	Capo	2657	197.8		199.7	
	+ AFB	4552	191.5	1.30%	193.7	1.81%
	+ FB	7699	192.0	2.94%	192.0	3.82%

Table III.4: Results for the PEKO instances. **Mode** indicates whether results are for original Capo, Capo with accelerated feedback (AFB), or regular feedback (FB). For all instances, we use the unconstrained feedback controller with 3 feedback iterations. We report the **best** and **average** HPWL results of 6 seeds. **CPU(s)** represents the total CPU time in seconds.

Circuit	**Mode**	**CPU**	**Best**	**Impr**	**Average**	**Impr**
Peko01	Capo	59	1.39		1.47	
	+ FB	240	1.36	2.23%	1.40	5.12%
Peko02	Capo	100	2.19		2.24	
	+ FB	404	2.06	6.26%	2.11	6.01%
Peko03	Capo	118	2.65		2.74	
	+ FB	514	2.49	6.27%	2.61	4.88%
Peko04	Capo	145	3.08		3.26	
	+ FB	583	2.91	5.47%	3.00	7.95%
Peko05	Capo	165	3.25		3.33	
	+ FB	671	3.07	5.36%	3.20	4.10%
Peko06	Capo	187	3.57		3.72	
	+ FB	692	3.38	5.47%	3.51	5.59%
Peko07	Capo	246	5.09		5.30	
	+ FB	975	4.89	4.26%	5.02	5.36%
Peko08	Capo	281	5.66		5.92	
	+ FB	1059	5.57	1.64%	5.76	2.81%
Peko09	Capo	349	6.39		6.78	
	+ FB	1280	6.09	4.73%	6.42	5.41%

Table III.5: Results for the PEKO instances. **Mode** indicates whether results are for original Capo (version 8.7), Capo with accelerated feedback (AFB), or regular feedback (FB). For all instances, we use the unconstrained feedback controller with 3 feedback iterations. We report the **best** and **average** HPWL results of 6 seeds. **CPU(s)** represents the total CPU time in seconds.

Circuit	**Mode**	**CPU**	**Best**	**Impr**	**Average**	**Impr**
Peko10	Capo	462	8.47		8.66	
	+ FB	1716	7.79	8.02%	8.00	7.59%
Peko11	Capo	453	8.32		8.89	
	+ FB	1754	8.12	2.43%	8.61	3.22%
Peko12	Capo	587	8.99		9.51	
	+ FB	2195	8.41	6.44%	8.60	9.60%
Peko13	Capo	683	11.04		11.04	
	+ FB	2611	9.85	5.12%	10.38	5.94%
Peko14	Capo	1169	16.49		17.41	
	+ FB	4278	15.48	6.10%	16.00	8.06%
Peko15	Capo	1465	21.17		22.26	
	+ FB	5531	20.21	4.50%	20.59	7.51%
Peko16	Capo	2051	22.77		22.95	
	+ FB	7348	22.51	1.15%	22.51	1.93%
Peko17	Capo	2141	25.07		25.25	
	+ FB	7995	24.57	2.04%	24.56	2.72%
Peko18	Capo	1434	24.66		25.12	
	+ FB	5063	23.47	4.82%	24.29	3.32%

Table III.6: Results for the IBM version 2 instances (easy and hard). **Mode** indicates whether the results represents original Capo's results or Capo with accelerated feedback (AFB). **CPU(s)** represents the total CPU time in seconds. For routability results, we report routing CPU time in seconds, number of routing violations, number of vias and routed wirelength (**WL**). **Impr** indicates the improvement percentage in wirelength for feedback over Capo. All placements were routed using the Linux version of Cadence's WarpRoute 2.4.

Circuit	**Mode**	**CPU(s)**	**#Viols**	**#Vias**	**WL**	**Impr**
ibm01e	Capo	10800	1238	147426	928179	
	+ AFB	6051	0	140353	872629	6.39%
ibm01h	Capo	10830	601	148416	909431	
	+ AFB	7964	103	146437	895000	1.54%
ibm02e	Capo	1411	0	311617	2371683	
	+ AFB	951	0	298034	2228700	6.75%
ibm02h	Capo	2138	0	308345	2240272	
	+ AFB	2365	0	309148	2222011	0.90%
ibm07e	Capo	2734	0	583716	4953781	
	+ AFB	1919	0	565161	4617930	6.86%
ibm07h	Capo	11501	450	611954	5124405	
	+ AFB	3160	0	575019	4657547	9.17%
ibm08e	Capo	1714	0	682384	5145286	
	+ AFB	1215	0	660089	4770640	7.75%
ibm08h	Capo	7560	59	726584	5213489	
	+ AFB	1180	0	669970	4841286	7.10%

Table III.7: Results for the IBM version 2 instances (easy and hard). **Mode** indicates whether the results represents original Capo's results or Capo with accelerated feedback (AFB). **CPU(s)** represents the total CPU time in seconds. For routability results, we report routing CPU time in seconds, number of routing violations, number of vias and routed wirelength (**WL**). **Impr** indicates the improvement percentage in wirelength for feedback over Capo. All placements were routed using the Linux version of Cadence's WarpRoute 2.4.

Circuit	**Mode**	**CPU(s)**	**#Viols**	**#Vias**	**WL**	**Impr**
ibm09e	Capo	2657	1	532641	3413247	
	+ AFB	3632	0	532894	3468914	-1.61%
ibm09h	Capo	2943	0	554942	3578004	
	+ AFB	4520	0	550386	3528053	1.40%
ibm10e	Capo	5556	0	852419	6831436	
	+ AFB	8218	0	849532	6970985	-2.03%
ibm10h	Capo	15835	0	941380	7771071	
	+ AFB	11242	0	896749	7266408	6.49%
ibm11e	Capo	3894	0	693248	5063947	
	+ AFB	5968	0	683467	5016880	0.92%
ibm11h	Capo	3973	0	715129	5213806	
	+ AFB	6165	0	704912	5178964	0.68%
ibm12e	Capo	33357	43	1107320	10408826	
	+ AFB	17018	0	1055410	10020246	3.65%
ibm12h	Capo	14541	1	1117458	10475534	
	+ AFB	17517	0	1077275	10086777	3.72%

Table III.8: Evaluating the impact of feedback using the ZCNT framework. All percentages are deviations from the pre-calculated reference, i.e, lower bounds to the suboptimality gap. All ZCNTs are applied to change the circuit statistics (as described in Subsection II.C) by 15%.

Circuit	**HPWL**		**Circuit**	**HPWL**	
	without feedback	with feedback		without feedback	with feedback
ibm01	19.03%	15.03%	ibm10	5.30%	3.30%
ibm02	8.05%	3.23%	ibm11	17.80%	4.95%
ibm03	6.38%	10.10%	ibm12	6.09%	2.97%
ibm04	4.56%	3.44%	ibm13	7.79%	4.15%
ibm05	2.90%	1.40%	ibm14	6.21%	3.88%
ibm06	6.07%	4.38%	ibm15	23.75%	5.16%
ibm07	5.76%	4.48%	ibm16	14.55%	4.84%
ibm08	7.05%	3.82%	ibm17	4.35%	3.27%
ibm09	11.44%	4.58%	ibm18	12.60%	11.95%

Chapter IV

New Approaches for Gene Chip Placement Benchmarking

As mentioned in the Introduction, there are two goals for this thesis: (1) quantifying the suboptimality of existing placement techniques, and (2) improving placement algorithms in case a significant suboptimality is proved. In the previous two chapters, we have already covered CMOS placement benchmarking and placement improvement. In this chapter and the next, we cover Gene chip placement benchmarking and placement improvement. Gene chip placement techniques are relatively newer than CMOS placement techniques. However, Gene chip techniques borrow from the rich literature of CMOS chips placement [46, 47, 50, 51, 48, 49].

This chapter starts by first describing previous placement approaches in Gene chips (Section IV.A), and then quantifying the suboptimality of these approaches (Section IV.B). We propose three techniques for suboptimality quantification, using lower bounds, instances with known optimal placements, and instances with known suboptimal placements. Our results indicate that Gene chip placement algorithms are suboptimal by a significant gap.

IV.A Previous Work on Border Length Minimization

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 [33], who proposed an optimal solution based on 2-dimensional Gray codes. Hannenhalli et al. [40] 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 [65].

Another generic approach is to use a metaheuristic, e.g. simulated annealing (SA) [63], to optimize the placement. In our SA implementation, we start by sorting the probes and threading them onto the chip. A square window is then slid over the chip. 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. After experimenting with various SA parameters, we chose to run SA with 6×6 windows with overlap of 3, with 6^3 iterations performed for every window position.

IV.B Benchmarking Approaches

The main contributions of this chapter are three new benchmarking methods. These include:

- calculating lower bounds for array design;
- constructing instances with known optimal border length; and
- constructing scaled instances with known suboptimal border length.

We describe each of these methods in detail in the following subsections.

IV.B.1 Calculating Lower Bounds for General Instances

In this subsection, we give lower bounds on the cost of optimum solutions for both the synchronous and asynchronous array design problems (SADP and AADP respectively) as defined in Subsection I.B.

We first start by giving lower bounds on the optimum solution for synchronous array design. 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 G_2 (as defined in Subsection I.B), we obtain the following:

Theorem IV.1 *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. Define 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 IV.1 we obtain:

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

We remark that 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 IV.1 gives one such example consisting of four dinucleotide probes, AC, GA, CT, and TG, which must be placed on a

Table IV.1: Total border cost, normalized border cost, gap from the synchronous placement lower bound, and CPU seconds (averages over 10 random instances) for the TSP+threading (TSP+1Thr), and the simulated annealing algorithm (SA).

Chip	LB	TSP+1Thr			SA		
Size	Cost	Cost	Gap(%)	CPU	Cost	Gap(%)	CPU
100	$0.41M$	$0.55M$	35.3	113	$0.58M$	42.4%	20769
200	$1.51M$	$2.14M$	41.6	1901	$2.42M$	59.9%	55658
300	$3.23M$	$4.67M$	44.3	12028	$5.50M$	70.2%	103668
500	$8.46M$	$12.70M$	50.1	109648	$15.43M$	82.5%	212390

2×2 grid. In this case, the lower bound on the number of conflicts is 8 while the optimum number of conflicts is 10.

We benchmark the previous approaches, TSP + threading and simulated annealing, using the proposed lower bounds for both the synchronous regime (Table IV.1) and the asynchronous regime (Table IV.2)[1]. Our results show a significant suboptimality gap for both regimes. In the synchronous case, demonstrated suboptimality gaps are of up to 50% for TSP + threading and of up to 82% for SA. An interesting trend is that performance tends to deteriorate as instance sizes increase. While the suboptimality results might be a consequence of loose lower bounds, the next section proves that this is unlikely.

IV.B.2 Constructing Instances with Known Optimum Solution

For CMOS placement, instances with known minimum-wirelength solutions may be constructed by "overlaying" signal nets within an already placed cell

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

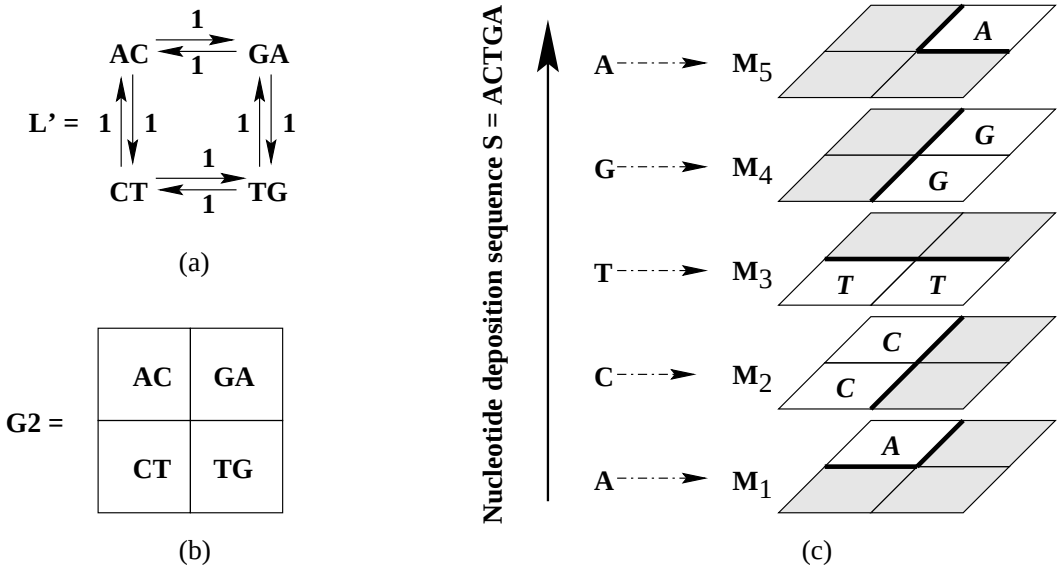


Figure IV.1: (a) Lower-bound digraph L' for the probes AC, GA, CT, and TG. The total arc weight of L' is 8. (b) Optimum two-dimensional placement of the probes. (c) Optimum embedding of the probes into the nucleotide deposition supersequence $S = ACTGA$. The optimum embedding has 10 conflicts, exceeding the lower bound by 2.

layout, such that each signal net has provably minimum length. This technique, proposed by Boese [37] and further explored by Chang, Cong, and Xie [19], 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 circuit netlist. 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 [33]. Our construction generates 4^k probes which are placeable such that every probe has border cost of 2 to each of its neighboring probes. This construction is illustrated in Figure IV.2.

Table IV.3 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). The

Table IV.2: Total border cost, normalized border cost, gap from the asynchronous pre-placement lower bound, and CPU seconds (averages over 10 random instances) for the TSP+threading (TSP+1Thr), and the simulated annealing algorithm (SA).

Chip	LB	TSP+1Thr			SA		
Size	Cost	Cost	Gap(%)	CPU	Cost	Gap(%)	CPU
100	$0.22M$	$0.44M$	99.5	113	$0.46M$	107.6	11713
200	$0.80M$	$1.72M$	115.8	1901	$1.84M$	130.9	42679
300	—	$3.80M$	—	12028	$4.16M$	—	101253
500	—	$10.43M$	—	$9.46M$	$11.57M$	—	222376

results confirm the suboptimality of existing placement methods noted in the previous subsection. On the other hand, 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.

IV.B.3 Constructing Instances with Known Suboptimal Solutions

Because constructed instances with known optimum solutions may not be representative of "real" instances, we also apply a technique by Hagen, Huang, and Kahng [37] 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

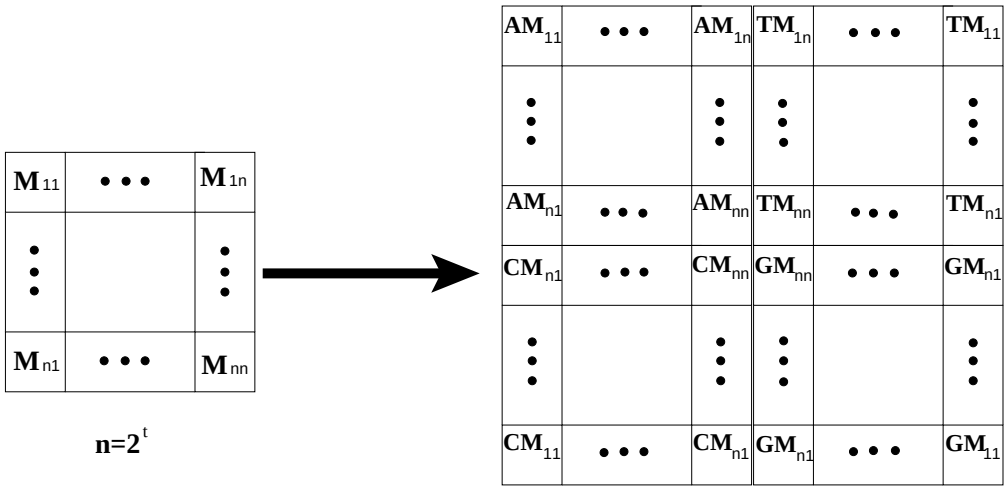


Figure IV.2: 2-dimensional Gray code placement.

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 IV.3. 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 the heuristic's solution quality would be said to scale well on this class of instances. Our attempts to test the scalability of TSP+Threading could not be included in the comparison due to unexpected crashes.

IV.C Conclusions

In this chapter we have studied the suboptimal behavior of Gene chip placement algorithms. We have proposed three benchmarking techniques based on calculating border length lower bounds and constructing gene chip instances with known optimal and suboptimal border length. Our techniques prove that current placement techniques have a significant suboptimality gap. Furthermore, there are strong indications of poor scaling behavior, given that suboptimality increases with

Table IV.3: Benchmarking the performance of placement algorithms for cases with known optimal number of conflicts.

Chip Size	Optimal Cost	TSP+Threading		SA	
		Cost	Gap(%)	Cost	Gap(%)
16×16	960	1380	44	1400	45
32×32	3968	6524	65	6298	58
64×64	16128	27072	68	24552	52
128×128	65024	111420	71	104832	61
256×256	261120	457100	75	405288	55
512×512	1046528	1844244	76	1706756	63

increasing chip sizes. We address this suboptimal behavior in the next chapter.

The material in this chapter is based on the following publications, where all authors' names are listed in alphabetical order. The dissertation author is the primary researcher and author.

- A. Kahng, I. Mandoiu, S. Reda, A. Zelikovsky and X. Xu, "Computer-Aided Optimization of DNA Array Design and Manufacturing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25(2), 2006, pp. 305 – 320.

- A. Kahng, I. Mandoiu, P. Pevzner, S. Reda and A. Zelikovsky, "Scalable Heuristics for Design of DNA Probe Arrays," *Journal of Computational Biology*, Vol. 11(2-3), 2004, pp. 429 – 447.

My coauthors (Prof. Andrew B. Kahng, Prof. Pavel Pevzner, Prof. Ion Mandoiu, Prof. Alex Zelikovsky, and Mr. Xu Xu) have all kindly approved the inclusion of the aforementioned publications in my thesis.

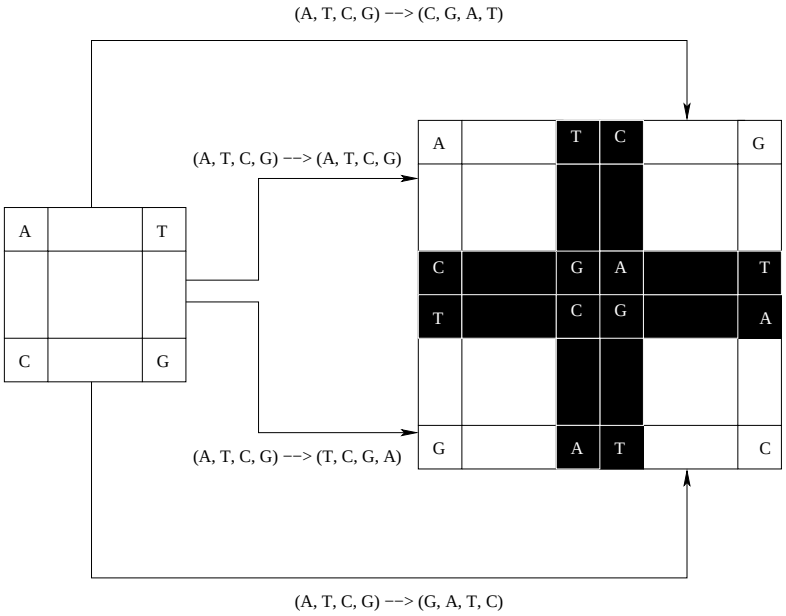


Figure IV.3: Scaling construction used in the suboptimality experiment.

Chapter V

New Placement Approaches for Gene Chips

In the previous chapter, we have proven that existing Gene chip placement algorithms suffer from a serious suboptimality gap. In this chapter, we propose a number of improved placement algorithms. We exploit the similarities between manufacturing processes for DNA arrays and VLSI CMOS chips, and demonstrate significant potential for transfer of electronic design automation methodologies [32, 79] to the newer DNA array design field. In this chapter, we propose three new placement heuristics. Specifically, in Subsection V.A we propose: a constructive epitaxial-based placement, an iterative improvement sliding window matching heuristic, and a constructive partitioning-based placement heuristic. We also propose optimal probe alignment algorithms in Section V.B. Our experimental results demonstrate significant placement improvements over previous approaches.

V.A New Scalable Placement Algorithms for Synchronous Array Design

In this subsection, we describe the engineering of near linear-time, yet high-quality, synchronous probe placement heuristics. A recurring motif in our

discussion is the value of technology transfer between the 40-year VLSI CMOS design literature and the newer field of DNA chip design. We point out analogies that inspire useful heuristics, as well as distinctions that hamper direct transfers. Two key design goals are (i) to enable scaling to 100 million or more placeable objects in the near future (say, within the current or next generation of workstations), and (ii) to enable easy parallelism (implying near-linear speedup on workstation clusters) wherever possible. We first describe an epitaxial growth algorithm inspired from the VLSI design literature and then describe a scalable version of it which we call row-epitaxial (*Reptx()*). Then, we give a highly scalable heuristic based on optimally re-placing an independent set of probes, again using placement improvement ideas from VLSI design. Finally, we give a constructive heuristic that is inspired by partitioning techniques in VLSI CMOS chips.

V.A.1 Epitaxial Growth Placement

In this section, we describe the so-called *epitaxial placement* approach to SADP and discuss some efficient implementation details. Epitaxial, or seeded crystal growth, placement is a technique that has been well-explored in the VLSI circuit placement literature [72, 78]. The technique essentially grows a two-dimensional placement around a single starting seed.

The algorithm in [40], which finds 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 V.1) 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. We have found that sites with more filled neighbors should have higher priority to be filled. In particular, we give highest priority to sites with 4 known neighbors. In the remaining cases we apply scaling coefficients to prioritize candidate probe-site pairs. In our implementation, if a probe is to be placed at a

site with $i < 4$ placed neighbors, then the average number of conflicts caused by this placement is multiplied by a coefficient $0 < k_i \leq 1$. Based on our experiments, we set $k_1 = 1$, $k_2 = 0.8$, and $k_3 = 0.6$. In our implementation we avoid repeated distance computations by keeping with each border site a list of probes sorted by normalized cost. For each site this list is computed at most four (and on the average two) times, i.e., when one of the neighboring sites is being filled while the site is still empty.

While the epitaxial algorithm achieves better results compared to other two-dimensional placements, it becomes impractical for DNA chips with dimensions of 300×300 or more. We make the following observation.

Observation 1: Any placement method can be trivially scaled by partitioning the set of probes and the probe array into K subsets ("chunks") each, then solving K independent placement problems. While runtime remains linear in the number of probes, two types of losses are incurred: (i) from lack of freedom of a probe to move anywhere other than its subset's assigned chunk of array sites, and (ii) lack of optimization on borders between chunks.

Based on Observation 1, we have implemented trivial scaling for the epitaxial algorithm using chunk sizes up to 50×50. However, the results are dominated by those obtained using the following scalable variant of the epitaxial algorithm, which we call the *row-epitaxial Reptx()* 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 row-major order, namely, row by row and within a row from left to right.

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

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 V.1: The Epitaxial 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, we cannot afford using any two-dimensional placement based on computing all pairwise distances between probes (such as TSP-based placement in [40]). Possible initial placement algorithms can be based on space-filling curve (e.g., Gray code) ordering [12]; indeed such orderings have had success in the VLSI context [10]. We found that excellent initial placements are obtained by simply ordering the probes lexicographically (this can be done in linear time by radix sort) and then threading them as in [40]. 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.

V.A.2 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 metaheuristics such as simulated annealing. More recently, strong neighborhood operators have been proposed which improve larger portions of the placement. For example, the DOMINO approach [27] iteratively determines an optimal reassignment of all objects within a given window of the placement. The end-case placer of [15] 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 [27] around the placement; cf. cycling and overlapping [41], row-ironing [15], etc.

For DNA arrays, an initial placement (and embedding) of probes in array sites may be improved by changing the placement and/or the embedding of individual probes. Guided by the VLSI experience and the intuition that randomly chosen pairs of optimally-embedded probes are extremely unlikely to be swappable with reduction in border cost, we focus on strong operators. We make the following observation.

Observation 2: Simultaneous probe re-placement of an entire window is not practical even for very small windows, but simultaneous probe re-placement of an independent set within a window is practical.

Observation 2 rules out direct analogs of the DOMINO [27] and end-case placer [15] VLSI placement approaches. Instead, we propose the following novel method of improving the placement solution within a small region of the array. While improvements are still possible, we choose a set of mutually non-

adjacent (independent) array cells, then re-place the probes in these cells according to a minimum-cost assignment, where the cost of assigning probe p to cell c is given by the sum of Hamming distances to the four neighbors. 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 [6, 81].

We have carefully studied a number of methods for choosing the independent set within a window: (i) random maximal independent set, (ii) chessboard based independent set (white squares or black squares), (iii) best result from among K different maximal independent sets, etc. We have found that using a single random maximal independent set ($K = 1$) yields the best tradeoff between solution quality and runtime. Therefore, we have implemented the above sliding window method as follows. (1) We first radix-sort all probes lexicographically and then perform 1-threading as in [40]. (2) Then, for each sliding $W_0 \times W_0$ window we choose one random maximal independent set of sites and determine the cost of (asynchronous) reassignment of each associated probe to each site, then reassign probes according to the minimum weight perfect matching in the resulting weighted bipartite graph. (3) The window slides in rows, beginning in the top-left corner of the array; at each step, it slides horizontally to the right as far as possible while maintaining a prescribed amount of *window overlap*. After the right side of the array is reached, the window returns to the left end of the next row while maintaining the prescribed overlap with the preceding row. When the bottom side of the array is reached, the window returns to the top-left corner. (4) The window-sliding continues until an entire pass through the array results in less than 0.1% reduction of border cost.[1]

Our experiments [47] have shown that an overlap equal to half the window size gives best results; we use this setting for all results reported in this chapter. Figure V.2 illustrates the heuristic tuning with respect to varying window sizes.

[1]Faster variants can restrict the number of such passes to a small constant, e.g., 5. For arrays with up to half a million probes our implementation makes around 20 passes.

We observe that larger window sizes lose out to smaller window sizes when CPU time is very limited. Unless otherwise noted, experimental results below are obtained with window size = 6 (i.e., 6×6) and window overlap = 3 (for these values, the typical size of the random maximal independent set is around 13). Other experiments have shown that more effort in each window (and fewer cycles) loses out to less effort in each window (and more cycles), i.e., being greedier within a single window thwarts overall solution quality. Specifically, using multiple iterations of independent-set matching within a given window, or choosing the best of several attempted independent-set matchings, worsens our results. Last, we emphasize that the Sliding-Window Matching algorithm is easily parallelizable after the initial (linear-time) sorting and 1-threading step; the previous methods of [40] do not have this property.

V.A.3 Partition-Based Probe Placement

In this section we propose a probe placement heuristic inspired from min-cut partitioning-based placement algorithms for VLSI circuits. Recursive partitioning has been the basis of numerous successful VLSI placement algorithms [11], [16], [92] 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 [34], often in a multilevel framework [16]. 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 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,

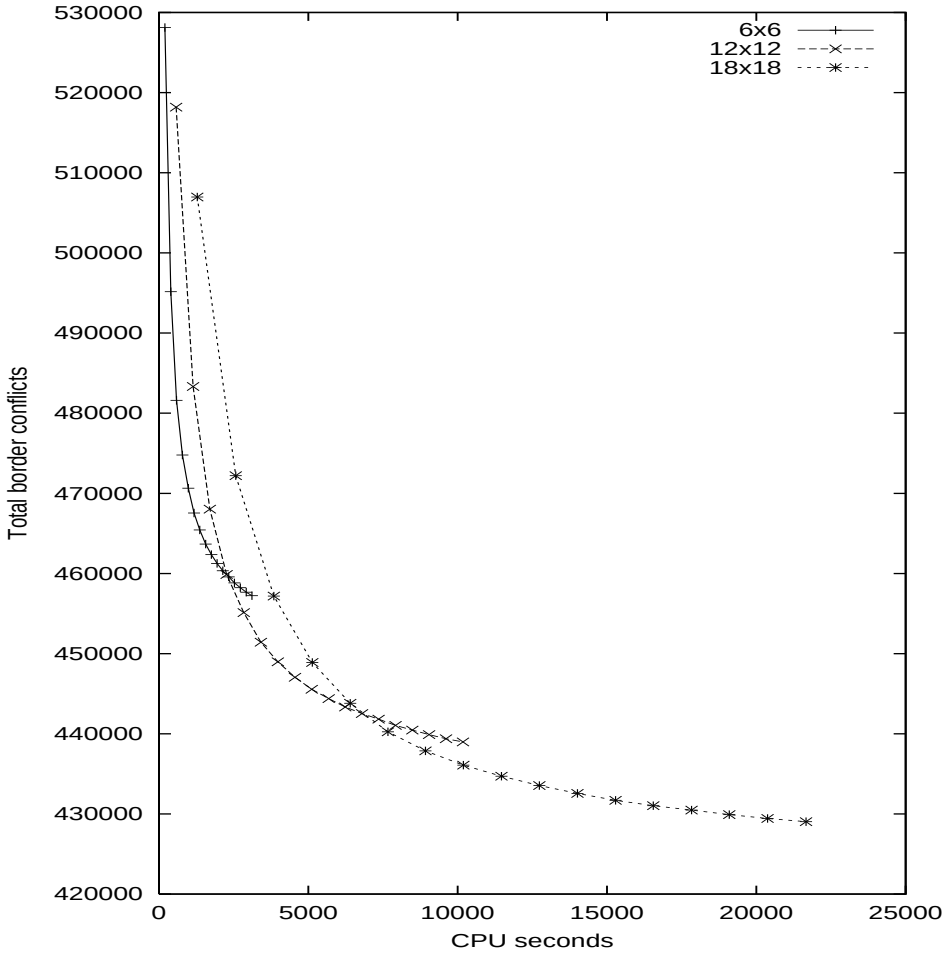


Figure V.2: Solution quality vs. runtime for Synchronous Sliding-Window Matching with varying window size; array size = 100×100.

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.[2] 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

[2]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.

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 V.3: The **SelectCentroid** procedure for selecting the centroid probes of subpartitions.

(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 V.3, is reminiscent of the k-center approach to clustering studied by Alpert et al. [9], and of methods used in large-scale document classification [25]. 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.

The complete partitioning-based placement algorithm for DNA arrays is given in Figure V.4. At a high level, our method resembles global-detailed approaches in the VLSI CAD literature [41], [75]. The algorithm recursively quadrisects every partition at a given level, assigning the probes so as to minimize distance to the centroids of subpartitions.[3] In the innermost of the three nested *for* loops of Figure V.4, we apply a multi-start heuristic, trying r different random probes as

[3]The variables i and j index the row and column of a given partition within the current level's array of 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 V.4: Partitioning-based DNA probe placement heuristic.

seed C_0 and using the result that minimizes total distance to the centroids. Once the maximum level L of the recursive partitioning is attained, detailed placement is executed via the row-epitaxial algorithm. Additional details and commentary are as follows.

- Within the innermost of the three nested *for* loops, our implementation actually performs, and benefits from, a *dynamic update* of the partition centroid whenever a probe is added into a given partition. Intuitively, this can lead to "elongated" rather than spheric clusters, but can also correct for unfortunate choices of the initial four centroids.[4]

- The straightforward implementation of *Reptx()*-based detailed placement within a given partition will treat the last locations within a region "unfairly", e.g., only one candidate probe will remain available for placing in a region's last location. To ensure that a uniform number of canidate probes for every position, our implementation permits "borrowing" probes from the next region in the *Reptx()* procedure. For every position of a region other than the last, we select the best probe from among at most m probes, where m is a pre-determined constant, in the current region *and the next*. (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.

V.B In-Place Optimization of Probe Embeddings

In our experience, we have found that separate optimization of probe placement and embedding yields better results for AADP than simultaneous op-

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

timization. For example, the asynchronous version of the epitaxial algorithm [46] and the asynchronous version of sliding-window matching [47] are both dominated by algorithms that work in two steps:

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 [40]).

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 begin with algorithms for optimally embedding a single probe with respect to its neighbors. We also propose and compare three methods for in-place optimization of probe embeddings, which we call the *Batched Greedy*, the *Chessboard*, and the *Sequential* algorithms.

V.B.1 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, our goal is to simultaneously align the given probe s to its embedded neighboring probes, while making sure this alignment gives a feasible embedding of s in the nucleotide deposition sequence S. In this section we give an efficient dynamic programming algorithm for computing this optimum alignment.

The Single Probe Alignment algorithm (see Figure V.5) essentially computes a shortest path in a specific directed acyclic graph $G = (V, E)$. Let p be the probe to be aligned, and let X be the set of already embedded probes adjacent to p. Each embedded probe $q \in X$ is a sequence of length $K = |S|$ over the alphabet $\{A, C, G, T, b\}$, with the j^{th} letter of q being either a blank or s_j, the j^{th} letter of the nucleotide deposition sequence S. The graph G (see Figure V.6) has vertex set $V = \{0, \ldots, k\} \times \{0, \ldots, K\}$ (where k is the length of the probe p and K is the

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

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

---

1. For each $j = 1, \dots, K$, 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 $i = 1, \dots, k$, $cost(i, 0) = \infty$.

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

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

For $i = 1, \dots, k$ 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 V.5: The single probe alignment algorithm.

length of the deposition sequence S), and edge set $E = E_{horiz} \cup E_{diag}$ where

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

and

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

The cost of a horizontal edge $(i, j-1) \to (i, j)$ is defined as the number of embedded probes in X which have a non-blank letter on j^{th} position, while the cost of a diagonal edge $(i-1, j-1) \to (i, j)$ is equal to the number of embedded probes of X with a blank on the j^{th} position. The Single Probe Alignment algorithm

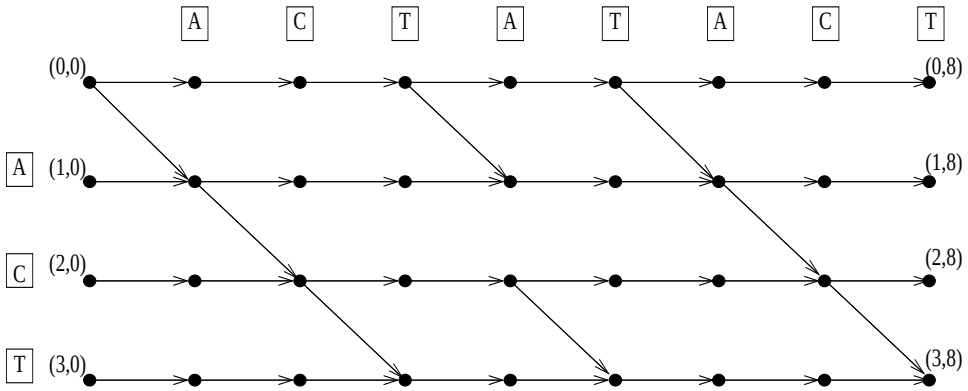


Figure V.6: Acyclic graph representing the dynamic programming computations of the single probe alignment algorithm.

computes the shortest path from the source node $(0, 0)$ to the sink node (k, K) using a topological traversal of G (the graph G is not explicitly constructed).

Theorem V.1 *The algorithm in Figure V.5 returns, in $O(kK)$ time, the minimum number of conflicts between an embedding of s and the adjacent embedded probes X (along with a minimum-conflict embedding of s).*

Proof : Each directed path from $(0, 0)$ to (k, K) in G consists of K edges, k of which must be diagonal. Each such path P corresponds to an embedding of p into S as follows. If the j^{th} arc of P is horizontal, the embedding has a blank in j^{th} position. Otherwise, the j^{th} arc must be of the form $(i - 1, j - 1) \rightarrow (i, j)$ 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.

Remark. 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 indegree of each node is at most 2^n. The weight of

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 V.7: The Batched Greedy algorithm.

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.

V.B.2 Algorithms for Iterative In-Place Embedding Optimization

Batched Greedy Optimization. We have implemented a natural greedy algorithm (GA) for optimizing probe embeddings. The GA finds a probe that offers largest cost gain from optimum re-embedding with respect to the (fixed) embeddings of its neighbors; the algorithm then implements this re-embedding, updates gains, and repeats. A faster *batched* version of GA (see Figure V.7) partially sacrifices its greedy nature in favor of runtime, via the mechanism of less-frequent gain updates. In other words, during a single *batched* phase we re-embed probes in greedy order according to the cost gains from re-embedding, but we do not update any gain while there are still independent probes with positive gain.

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 V.8: The Chessboard algorithm.

Chessboard Optimization. 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 V.8) 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 Subsection V.B.1) with $n = 2$ it alternatively optimizes the black and white 2×1 tiles.

Sequential Optimization. In this method, we perform optimal re-embedding of probes in a sequential row-by-row fashion. 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.

V.C Empirical Evaluation

In this section we compare several probe placement heuristics: the TSP 1-threading heuristic of [40] (TSP+1Thr), the Row-Epitaxial and sliding-window matching (SWM) heuristics of [47], a simulated annealing algorithm (SA) and the recursive partitioning based algorithm (RPART).[5] We used an upper bound of 20000 on the number of candidate probes in Row-Epitaxial, and 6×6 windows with overlap 3 for SWM.

Table V.1 gives the results produced by the TSP+1Thr, Row-Epitaxial, SWM, and SA heuristics on random instances with chip sizes between 100 and 500 and probe length equal to 25. Among the four heuristics, Row-Epitaxial is the algorithm with highest solution quality (i.e., lowest border cost), while SWM is the fastest, offering competitive solution quality with much less runtime. SA takes the largest amount of time, and also gives the worse solution quality. Although it may be possible to improve SA convergence speed by extensive fine-tuning of its various parameters, we expect that SA results will always remain dominated by those of the other heuristics.

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

We next discuss in more detail the runtime of RPART, which is some-

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

Table V.1: Total border cost ($\times 10^6$), gap from the synchronous placement lower bound, and CPU seconds (averages over 10 random instances) for the TSP heuristic (TSP+1Thr), the row-epitaxial (Row-Epitaxial) heuristic, sliding-window matching (SWM) heuristic, and the simulated annealing algorithm (SA).

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

Table V.2: Total border cost ($\times 10^6$) and CPU seconds (averages over 10 random instances) for the recursive partitioning algorithm with recursion depth $L = 2$ and number of restarts r varying from 1 to 1000.

Chip	RPART $r = 1$ $L = 2$			RPART $r = 10$ $L = 2$			RPART $r = 100$ $L = 2$			RPART $r = 1000$ $L = 2$		
Size	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU
100	0.49	19.5	22	0.49	19.5	24	0.49	19.5	47	0.49	19.5	238
200	1.86	23.1	276	1.86	23.1	283	1.86	23.1	394	1.86	23.1	1064
300	4.07	26.0	1501	4.07	26.0	1527	4.07	26.0	1769	4.07	26.0	4231
500	11.06	30.7	9531	11.05	30.6	9678	11.04	30.6	13158	11.04	30.6	17014

how unusual for a recursive partitioning algorithm in that it may get smaller with an increase in recursion depth. Let the number of probes in a chip be n. The two main contributors to RPART runtime are the recursive partitioning phase, whereby the probes are divided into smaller and smaller partition regions, and the detailed placement step which is achieved by running Row-Epitaxial within each partition region (with borrowing from next region when needed). Since executing the procedure *SelectCentroid()* and distributing all the probes in a partition region to its four sub-regions takes time proportional to the number of probes in a region, the total runtime for each recursion depth is $O(n)$, and the overall runtime for the recursive partitioning phase is $O(Ln)$. Clearly, this component of the runtime increases linearly with the recursion depth. On the other hand, in our implementation of RPART, the Row-Epitaxial algorithm used in detailed placement considers at most $\min\{m, 2n/4^L\}$ candidate probes for placement at any given position ($m = 20000$ is a pre-determined upper-bound which we impose based on the empirical results in [47], and $2n/4^L$ is a bound that follows from the fact that we never consider candidates from more than two consecutive lowest-level partition regions). Thus, the total time needed by the detailed placement step is $O(n \min\{m, 2n/4^L\})$, which will decrease with increasing L once L exceeds $\log_4(2n/m)$. This explains why the overall RPART runtime in Table V.3 decreases with increasing L, and also explains why solution quality may slightly degrade with increasing L due to the reduced number of probe candidates considered by Row-Epitaxial for each chip location.

V.C.1 Comparison of Complete Probe Placement and Embedding Flows

Table V.4 compares the new probe embedding algorithm with Batched Greedy and Chessboard on random instances with chip sizes between 100 and 500 and probe length 25 for which the 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.

Table V.3: Total border cost ($\times 10^6$), gap from the synchronous placement lower bound, and CPU seconds (averages over 10 random instances) for the recursive partitioning algorithm with recursion depth varying between 1 and 3.

Chip	Lower	RPART			RPART			RPART		
	Bound	$r = 10$ $L = 1$			$r = 10$ $L = 2$			$r = 10$ $L = 3$		
Size	Cost	Cost	Gap(%)	CPU	Cost	Gap(%)	CPU	Cost	Gap(%)	CPU
100	0.41	0.47	16.1	69	0.49	20.0	24	0.50	23.1	10
200	1.51	1.81	19.9	992	1.86	23.4	283	1.92	27.2	81
300	3.23	4.13	27.9	3529	4.07	26.0	1527	4.17	29.1	240
500	8.46	11.28	33.4	10591	11.05	30.6	9678	11.13	31.6	3321

The results show that re-embedding of the probes in a sequential row-by-row order leads to reduced border cost with similar runtime compared to previous methods.

In another series of experiments, we ran complete placement and embedding flows obtained by combining each of the five two-dimensional placement algorithms with the sequential in-place re-embedding algorithm. Results are given in Tables V.5 and V.6. Again, SA and TSP+1Thr are dominated by both REPTX and SWM in both conflict cost and running time. REPTX produces less conflicts than SWM but SWM is considerably faster. Recursive partitioning consistently outperforms the best previous flow (row-epitaxial + sequential re-embedding) – by an average of 4.0% – with similar or lower runtime.

We have also validated our methods on the set of probes for an Affymetrix Humane Genome chip. This 712×712 DNA chip is filled by pairs of SNP's (or single nucleotide polymorphism where each consists of the original probe and a copy with the middle nucleotide changed) and a small amount of control probes ($< 1\%$) each having a pre-determined placement. The (truncated) periodic nucleotide deposition sequence used by Affymetrix (and in our experiments) has length 74: this sequence is sufficiently long to accommodate all probes and cheaper than the universal 100-long nucleotide sequence by 26%. The flow that gave the best

Table V.4: Total border cost ($\times 10^6$), gap from the asynchronous post-placement lower bound, and CPU seconds (averages over 10 random instances) for the batched greedy, chessboard, and sequential in-place re-embedding algorithms.

Chip	Lower	Batched			Chessboard			Sequential		
	Bound	Greedy								
Size	Cost	Cost	Gap(%)	CPU	Cost	Gap(%)	CPU	Cost	Gap(%)	CPU
100	0.36	0.46	25.7	40	0.44	20.5	54	0.44	19.9	64
200	1.42	1.80	26.3	154	1.72	20.9	221	1.71	20.3	266
300	3.13	3.96	26.7	357	3.80	21.5	522	3.77	20.6	577
500	8.59	10.92	27.1	943	10.43	21.4	1423	10.38	20.9	1535

Table V.5: Total border cost ($\times 10^6$), gap from the asynchronous pre-placement lower bound, and CPU seconds (averages over 10 random instances) for the TSP heuristic of [26] (TSP+1Thr), the row-epitaxial (Row-Epitaxial) and sliding-window matching (SWM) heuristics of [35], and the simulated annealing algorithm (SA).

Chip	LB	TSP+1Thr			Row-Epitaxial			SWM			SA		
Size	Cost	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU
100	0.22	0.44	99.5	113	0.42	88.3	161	0.44	99.8	93	0.46	107.6	11713
200	0.80	1.72	115.8	1901	1.61	101.4	1368	1.72	115.6	380	1.84	130.9	42679
300	—	3.80	—	12028	3.53	—	3861	3.80	—	861	4.16	—	101253
500	—	10.43	—	$10.96M$	9.46	—	12044	10.16	—	2239	11.57	—	222376

Table V.6: Total border cost ($\times 10^6$), gap from the asynchronous pre-placement lower bound, and CPU seconds (averages over 10 random instances) for the recursive partitioning algorithm followed by sequential in-place re-embedding.

Chip	Lower	RPART			RPART			RPART		
	Bound	$r = 10$ $L = 1$			$r = 10$ $L = 2$			$r = 10$ $L = 3$		
Size	Cost	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU	Cost	Gap (%)	CPU
100	0.22	0.39	78.3	123	0.40	81.1	44	0.41	86.2	10
200	0.80	1.52	90.9	1204	1.55	93.5	365	1.57	97.0	101
300	—	3.49	—	3742	3.41	—	1951	3.43	—	527
500	—	9.54	—	11236	9.35	—	10417	9.30	—	3689

results consists of the following steps: (1) Probe embedding using a SNP-aware version of "earliest possible" embedding; (2) Lexicographical sorting of the embedded probes followed by 1-threading; (3) Synchronous sliding window matching – we used 48×48 windows with overlap 24 – where synchronous here refers to computing Hamming distances between embedded probes rather than un-embedded probes; (4) Row-epitaxial with $k_0 = 80$ rows of lookahead; and (5) A SNP-aware version of the sequential alignment algorithm. The final number of conflicts was reduced by 4.2% with respect to the Affymetrix optimized placement.[6]

V.C.2 Results on Instances with Known Optimal and Sub-optimal Border Length

Table V.7 gives results from executing various placement heuristics on the instances with known border length. The results show improved behavior

[6]We understand [42] that Affymetrix uses placement techniques that are similar to row-epitaxial placement combined with earliest possible alignment. This probably explains why our improvement is relatively small. Furthermore, we note that the reduction in number of masks from 100 to 74 also reduces somehow the freedom that can be exploited by our dynamic programming alignment algorithms.

Table V.7: Comparison of placement algorithm performance for cases with known optimal conflicts. SWM uses a window size of 20×20 and a step of 10. Row-epitaxial uses $10000/chipsize$ rows of lookahead.

Chip	Optimal	TSP+ Threading		Row-Epitaxial		SWM		RPART	
Size	Cost	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

compared to previous approaches (TSP+threading). We reduce the suboptimal gap by around 40%. We also observe that the SWM heuristic is performing better than the row-epitaxial heuristic. Table V.8 shows results from executing the various placement heuristics on scaled versions of random DNA probe sets, with the original instances ranging in size from 100×100 to 500×500, and the scaled instances thus ranging in size from 200×200 to 1000×1000. This table shows that in general, placement algorithms for DNA arrays offer excellent scaling suboptimality. We believe that this is primarily due to the already noted fact that algorithm quality 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.

Table V.8: Comparison of placement algorithm suboptimality for various benchmarks. Each entry represents both the upper bound and the actual placement result after scaling. SWM uses a window size of 20×20 and a step of 10. Row epitaxial uses $10000/chipsize$ lookahead rows.

Instance	Row			SWM			RPART		
Instance	Epitaxial								
Size	U-Bound	Actual	Ratio	U-Bound	Actual	Ratio	U-Bound	Actual	Ratio
100	2.02	1.48	0.73	2.20	1.99	0.91	1.92	1.43	0.73
200	7.70	6.38	0.83	8.48	6.88	0.81	7.49	6.11	0.82
300	16.82	12.79	0.76	18.64	13.96	0.75	16.69	12.56	0.75
400	29.24	24.62	0.84	32.54	26.84	0.82	30.45	24.24	0.80
500	44.89	38.14	0.85	49.80	41.85	0.84	47.33	37.81	0.80

V.C.3 Improved Integration of Probe Placement and Embedding

As noted in [46], allowing arbitrary, or asynchronous, embeddings leads to further reductions in border length compared to synchronous embedding (e.g., contrast (b) and (c) in Figure V.9). An interesting question is finding the best order in which the placement and embedding degrees of freedom should be exploited.

Our earlier method performs 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.

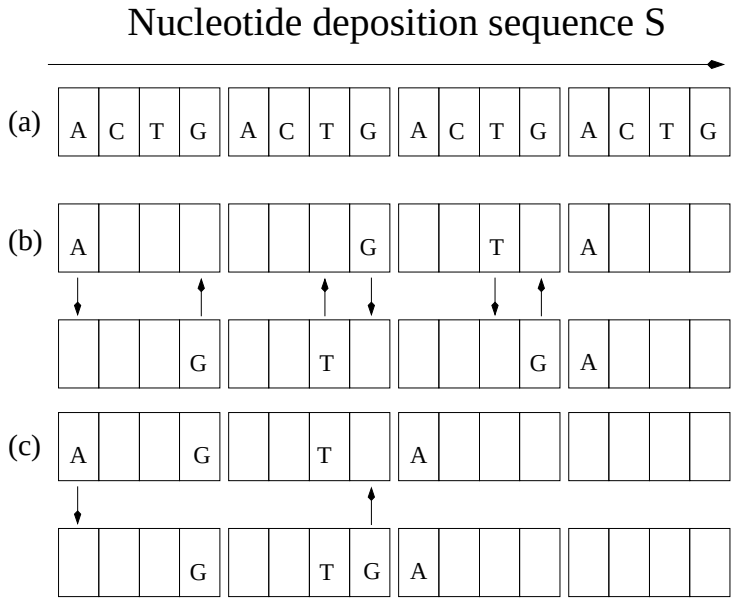


Figure V.9: (a) Periodic deposition sequence. (b) Synchronous embedding of the probes $AGTA$ and $GTGA$ gives 6 border conflicts (indicated by arrows). (c) "As soon as possible" asynchronous embedding of the probes $AGTA$ and $GTGA$ gives only 2 border conflicts.

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

- 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 current embodiment of the modified scheme, we implement the first step by using for each probe the "as soon as possible," or *ASAP*, embedding (see Figure V.9(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.

To empirically evaluate the advantages of ASAP embedding, we compare it against the synchronous initial embedding method on test cases ranging in size from 100×100 to 500×500. For both methods, the second and third steps are implemented using row epitaxial and sequential in-place probe re-embedding algorithms discussed earlier.

Tables V.9 and V.10 give the border length and CPU time (in seconds) for the two methods. Each number in these tables represents the average over 10 test cases of the given size. Surprisingly, the simple switch from synchronous to ASAP initial embedding results in 5-7% reduction in total border-length. Furthermore, the runtimes for the two methods are comparable. In fact, sequential re-embedding becomes faster in the ASAP-based method compared to the synchronous-based one since fewer iterations are needed to converge to a locally optimal solution (the number of iterations drops from 9 to 3 on the average).

V.D Conclusions

In this chapter, we have addressed the suboptimality of existing placement algorithms for Gene chips. We have shown that significant reductions in border length can be obtained by drawing on algorithmic techniques developed in the field of VLSI CMOS design automation.

We conclude with some remarks on the similarities and differences between VLSI physical design and physical design for DNA arrays. First, while

Table V.9: Total border cost (averages over 10 random instances) for synchronous and ASAP initial probe embedding followed by row-epitaxial and iterated sequential in-place probe re-embedding.

Chip	Synchronous Initial Embedding			ASAP Initial Embedding			Impr.
	Sync Embed	REPTX	Re-Embed	ASAP Embed	REPTX	Re-Embed	(%)
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 V.10: CPU seconds (averages over 10 random instances) for synchronous and ASAP initial probe embedding followed by row-epitaxial and iterated sequential in-place probe re-embedding.

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

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 scaled border cost when the number of probes in the design grows. Second, the lower bounds for DNA probe placement and embedding appear to be tighter than those available in the VLSI placement literature. Developing even tighter lower bounds is, of course, an important open problem.

The material in this chapter is based on the following publications, where all authors' names are listed in alphabetical order. The dissertation author is the primary researcher and author.

- A. Kahng, I. Mandoiu, S. Reda, A. Zelikovsky and X. Xu, "Computer-Aided Optimization of DNA Array Design and Manufacturing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25(2), 2006, pp. 305 – 320.

- A. Kahng, I. Mandoiu, P. Pevzner, S. Reda and A. Zelikovsky, "Scalable Heuristics for Design of DNA Probe Arrays," *Journal of Computational Biology*, Vol. 11(2-3), 2004, pp. 429 – 447.

My coauthors (Prof. Andrew B. Kahng, Prof. Pavel Pevzner, Prof. Ion Mandoiu, Prof. Alex Zelikovsky, and Mr. Xu Xu) have all kindly approved the inclusion of the aforementioned publications in my thesis.

Chapter VI

Conclusions

At the beginning of this thesis, we posed two critical questions for CMOS and Gene chips:

1. Are placement algorithms optimal (or "close" to) optimal on realistic chips?
2. If the previous question is answered negatively, how can placement algorithms be improved?

We now review the progress this thesis has made toward answering these two questions.

In the context of VLSI CMOS chips, we have proposed a new technique, zero-change netlist transformations, that answers Question (1) negatively. The main merits for the proposed approach are as follows.

- ZCNT proves, for the the first time, that placers are significantly suboptimal (up to 22%) on circuit instances that are realistic.
- ZCNT gives lower bounds to the suboptimality gaps for realistic synthesized circuits from any arbitrary given circuit.
- ZCNT gives the exact entire suboptimality gap for special constructed circuits.
- ZCNT is the only known technique to quantify the suboptimality gap for Steiner wirelength metrics.

- ZCNT gives the ability to test the consistency of placers by using any given circuit as a "seed" to create more benchmark circuits.

For future work, ZCNT can be used to improve the placability of circuits using hyperedge decomposition. We have explored that direction in the thesis, but we believe there are more possible improvements. As for the benchmarking problem in general, it is certainly far from being solved – we need mechanisms to calculate tight bounds on the suboptimality gap of any given circuit.

In response to the negative answer obtained from the ZCNT framework, we have pursued improving placement techniques per Question (2). We have proposed placement feedback as a simple, viable way to improve top-down placement. Feedback accurately accounts for terminal propagation between different placement subproblems, and leads to an improvement in total wirelength. We have integrated the proposed approach in the open-source placer Capo, and we have demonstrated its effectiveness on regular benchmarks and using the ZCNT benchmarking framework. We have obtained up to 15% improvements in wirelength.

In the context of Gene chips, we have answered Question (1) negatively. We have proposed three new benchmarking techniques that

- calculate new lower bounds for the border length of any given chip;
- construct chip instances with known optimal placements; and
- construct scaled chip instances with known suboptimal placements.

Our empirical results show that existing Gene chip placement algorithms have a substantial suboptimality gap of about 42%. To reduce the suboptimality in response to Question (2), we have proposed a wealth of new placement techniques:

- epitaxial placement as a fast constructive heuristic;
- scalable matching-based placement as an iterative improvement heuristic; and
- partitioning-based placement as an alternative constructive-based heuristic.

We have also devised embedding algorithms to exploit probe placement in a 3-dimensional setting. Using the devised constructive, iterative, and embedding heuristics, we have proposed a physical design flow for Gene chips. Our flow improves the placements of academic chips by 18% and an industrial Human Genome chip by 6%. A possible direction of future research is to find formulations and methods for integrated optimization of Gene chip test structure design and physical design. Since test structures are typically pre-placed at sites uniformly distributed across the array, integrated optimization can have a significant impact on the total placement quality.

Bibliography

[1] "http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Capo," *GSRC Bookshelf*.

[2] "http://www.affymetrix.com."

[3] S. Adya, I. Markov, and P. Villarrubia, "On Whitespace and Stability in Mixed-Size Placement," in *Proc. IEEE International Conference on Computer Aided Design*, 2003, pp. 311–318.

[4] S. N. Adya, M. C. Yildiz, I. L. Markov, P. G. Villarrubia, P. N. Parakh, and P. H. Madden, "Benchmarking for Large-Scale Placement and Beyond," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2003, pp. 95–103.

[5] A. Agnihotri, M. Yildiz, A. Khatkhate, A. Mathur, S. Ono, and P. Madden, "Fractional Cut: Improved Recursive Bisection Placement," in *Proc. IEEE International Conference on Computer Aided Design*, 2003, pp. 307–310.

[6] S. Akers, "On the Use of the Linear Assignment Algorithm in Module Placement," in *Proc. ACM/IEEE Design Automation Conference*, 1981, pp. 13–144.

[7] C. Alpert, A. B. Kahng, G.-J. Nam, S. Reda, and P. Villarubia, "A Fast Hierarchical Quadratic Placement Algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25(4), 2006, pp. 678–691.

[8] C. J. Alpert, J. H. Huang, and A. B. Kahng, "Multilevel Circuit Partitioning," in *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 530–533.

[9] C. J. Alpert and A. B. Kahng, "Geometric Embeddings for Faster (and Better) Multi-Way Netlist Partitioning," in *Proc. ACM/IEEE Design Automation Conference*, 1993, pp. 743–748.

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

[11] C. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A survey," *Integration: The VLSI Journal*, vol. 19, 1995, pp. 1–81.

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

[13] J. Beardwood, J. Halton, and J. Hammersley, "The Shortest Path through Many Points," *Proc. Cambridge Philos. Soc. 55*, 1959, pp. 299–327.

[14] M. A. Breuer, "Min-Cut Placement," *Design Automation and Fault Tolerant Computing*, vol. 1(4), 1977, pp. 343–362.

[15] A. Caldwell, A. B. Kahng, and I. Markov, "End-Case Placers for Standard-Cell Layout," in *Proc. ACM/IEEE International Symposium on Physical Design*, 1999, pp. 90–96.

[16] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Can Recursive Bisection Alone Produce Routable Placements?" in *Proc. ACM/IEEE Design Automation Conference*, 2000, pp. 477–482.

[17] A. E. Caldwell, A. B. Kahng, and I. L. Markov, "Optimal Partitioners and End-case Placers for Standard-Cell Layout," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19(11), 2000, pp. 1304–1313.

[18] C. Chang, J. Cong, M. Romesis, and M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2004, vol. 23(4), pp. 537–549.

[19] C. Chang, J. Cong, and M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms," in *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2003, pp. 621–627.

[20] C. E. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," in *Proc. IEEE International Conference on Computer Aided Design*, 1994, pp. 690–695.

[21] C. K. Cheng and E. S. Kuh, "Module Placement Based on Resistive Network Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4, 1984, pp. 115–122.

[22] J. Cong, T. Kong, J. R. Shinnerl, M. Xie, and X. Yuan, "Large-Scale Circuit Placement: Gap and Promise," in *Proc. IEEE International Conference on Computer Aided Design*, 2003, pp. 883–890.

[23] J. Cong, M. Romesis, and M. Xie, "Optimality and Scalability Study of Partitioning and Placement Algorithms," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2003, pp. 88–94.

[24] J. Cong, J. R. Shinnerl, M. Xie, T. Kong, and X. Yuan, "Large-Scale Circuit Placement," *ACM Transactions on Design Automation of Electronic Systems*, vol. 10(2), 2005, pp. 389–430.

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

[26] D. Cyganski, R. Vaz, and V. Virball, "Quadratic Assignment Problems Generated with the Palubetskis Algorithm are Degenerate," *IEEE Transactions on Circuits and Systems – I: Fundamental Theory and Applications*, vol. 41(7), 1994, pp. 481–484.

[27] K. Doll, F. Johannes, and K. Antreich, "Iterative Placement Improvement by Network Flow Methods," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13(10), 1994, pp. 1189–1200.

[28] W. E. Donath, "Statistical Properties of the Placement of a Graph," *SIAM J. Appl. Math*, vol. 16(2), 1968, pp. 439–457.

[29] R. Dorf and R. Bishop, *Modern Control Systems*, Ninth edition. Prentice Hall, 2000.

[30] A. E. Dunlop and B. W. Kernighan, "A Procedure for Placement of Standard-Cell VLSI Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 4(1), 1985, pp. 92–98.

[31] H. Eisenmann and F. M. Johannes, "Generic Global Placement and Floorplanning," in *Proc. ACM/IEEE Design Automation Conference*, 1998, pp. 269–274.

[32] J. J. Engel *et al.*, "Design methodology for IBM ASIC products," *IBM Journal for Research and Development*, vol. 40(4), 1996, p. 387.

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

[34] C. M. Fiduccia and R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," in *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 175–181.

[35] 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.

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

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

[38] K. M. Hall, "An r-Dimensional Quadratic Placement Algorithm," *Management Science*, 1970, vol. 17, pp. 219–229.

[39] M. Hanan and J. M. Kurtzberg, "Placement Techniques," in *Design Automation of Digital Systems (M. A. Breuer Ed.)*, 1972, pp. 213–282.

[40] S. Hannenhalli, E. Hubbell, R. Lipshutz, and P. A. Pevzner, "Combinatorial Algorithms for Design of DNA Arrays," *Chip Technology (J. Hoheisel Ed.)*, 2002.

[41] D. J.-H. Huang and A. B. Kahng, "Partitioning-Based Standard-Cell Global Placement With an Exact Objective," in *Proc. ACM/IEEE International Symposium on Physical Design*, 1997, pp. 18–25.

[42] E. Hubbell and M. Mittman, personal communication, 2002.

[43] A. B. Kahng and S. Mantik, "On Mismatches between Incremental Optimizers and Instance Perturbations in Physical Design Tools," in *Proc. IEEE International Conference on Computer Aided Design*, 2000, pp. 17–22.

[44] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools," in *International Symposium on Quality in Electronic Design*, 2002, pp. 206–211.

[45] A. B. Kahng, I. Markov, and S. Reda, "On Legalization of Row-Based Placements," in *Proc. IEEE Great Lakes Symposium on VLSI*, 2004, pp. 214–219.

[46] A. B. Kahng, I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky, "Border Length Minimization in DNA Array Design," in *Proc. 2nd International Workshop on Algorithms in Bioinformatics*, 2002, pp. 435–448.

[47] A. B. Kahng, I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky, "Engineering a Scalable Placement Heuristic for DNA Probe Arrays," in *Proc. 7th Annual International Conference on Research in Computational Molecular Biology*, 2003, pp. 148–156.

[48] A. B. Kahng, I. Măndoiu, P. Pevzner, S. Reda, and A. Zelikovsky, "Scalable heuristics for design of DNA probe arrays," *Journal of Computational Biology*, 2004, pp. 429–447.

[49] A. B. Kahng, I. Măndoiu, S. Reda, A. Zelikovsky, and X. Xu, "Computer-Aided Optimization of DNA Array Design and Manufacturing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25(2), 2006, pp. 305–320.

[50] A. B. Kahng, I. Măndoiu, S. Reda, X. Xu, and A. Zelikovsky, "Design Flow Enhancements for DNA Arrays," in *International Conference on Computer Design*, 2003, pp. 116–123.

[51] A. B. Kahng, I. Măndoiu, S. Reda, X. Xu, and A. Zelikovsky, "Evaluation of Placement Techniques for DNA Probe Array Layout," in *Proc. IEEE International Conference on Computer Aided Design*, 2003, pp. 262–269.

[52] A. B. Kahng and S. Reda, "Placement Feedback: A Concept and Method for Better Min-Cut Placement," in *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 357–362.

[53] A. B. Kahng and S. Reda, "Evaluation of Placer Suboptimality Via Zero-Change Netlist Transformations," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2005, pp. 208–215.

[54] A. B. Kahng and S. Reda, "Wirelength Minimization for Min-Cut Placements via Placement Feedback," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, to appear.

[55] A. B. Kahng and S. Reda, "Zero-Change Netlist Transformations: A New Technique for Placement Benchmarking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, to appear.

[56] A. B. Kahng, S. Reda, and Q. Wang, "Architecture and Details of a High Quality, Large-Scale Analytical Placer," in *Proc. IEEE International Conference on Computer Aided Design*, 2005, pp. 891–898.

[57] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 11(7), 1992, pp. 893–902.

[58] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytical Placer," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2004, pp. 18–25.

[59] G. Karypis, R. Aggarwal, V. Kumar, and S. Shekhar, "Multilevel Hypergraph Partitioning: Application in VLSI Domain," in *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 526–529.

[60] G. Karypis and V. Kumar, "Multilevel k-way Hypergraph Partitioning," in *Proc. ACM/IEEE Design Automation Conference*, 1999, pp. 343–348.

[61] A. A. Kennings and I. L. Markov, "Analytical Minimization of Half-Perimeter Wirelength," in *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2000, pp. 179–184.

[62] B. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," *The Bell Systems Technical Journal*, vol. 49(2), 1970, pp. 291–307.

[63] S. Kirkpatrick, C. G. Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220(4568), 1983, pp. 671–680.

[64] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, "GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10(3), 1991, pp. 356–365.

[65] T. Kozawa *et al.*, "Automatic Placement Algorithms for High Packging Density VLSI," in *Proc. ACM/IEEE Design Automation Conference*, 1983, pp. 175–181.

[66] B. Landman and R. Russo, "On a Pin Versus Block Relationship for Partitions of Logical Graphs," *IEEE Transactions on Computers*, vol. 20(C), 1971, pp. 793–813.

[67] C. Li *et al.*, "Routability-Driven Placement and White Space Allocation," in *Proc. IEEE International Conference on Computer Aided Design*, 2004, pp. 394–401.

[68] R. Lipshutz, S. Fodor, T. Gingeras, and D. Lockhart, "High Density Synthetic Oligonucleotide Arrays," *Nature Genetics*, vol. 21, 1999, pp. 20–24.

[69] Q. Liu and M. Marek-Sadowska, "A Study of Netlist Structure and Placement Efficiency," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2004, pp. 198–203.

[70] W. Naylor, "Non-Linear Optimization System and Method for Wirelength and Density Within an Automatic Electronic Circuit Placer," *US Patent 6282693*, 2001.

[71] S. Ono and P. Madden, "On Structure and Suboptimality in Placement," in *Proc. IEEE Asia and South Pacific Design Automation Conference*, 2005, pp. 331–336.

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

[73] M. Queyranne, "Performance Ratio of Polynomial Heuristics for Triangle Inequality Quadratic Assignment Problem," *Operations Research Letters*, vol. 4, 1986, pp. 231–342.

[74] S. Sahni and T. Gonzalez, "P-Complete approximation problems," *Journal of the ACM*, vol. 23, 1976, pp. 555–565.

[75] M. Sarrafzadeh and M. Wang, "NRG: Global and Detailed Placement," in *Proc. IEEE International Conference on Computer Aided Design*, 1997, pp. 532–537.

[76] C. Sechen and K. W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," in *Proc. IEEE International Conference on Computer Aided Design*, 1987, pp. 478–481.

[77] C. Sechen and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," in *Proc. ACM/IEEE Design Automation Conference*, 1986, pp. 432–439.

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

[79] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 3rd ed., Kluwer Academic Publishers, 1999, p. 257.

[80] J. Steele, "Subadditive Euclidean Functionals and Non-Linear Growth in Geometric Probability," *Ann. Probability*, vol. 9, 1981, pp. 365–376.

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

[82] D. Stroobandt, J. Depreitere, and J. V. Campenhout, "Generating New Benchmark Designs Using a Multi-Terminal Net Model," *Integration, the VLSI Journal*, vol. 27(2), 1999, pp. 113–129.

[83] D. Stroobandt and F. J. Kurdahi, "In the Characterization of Multi-point Nets in Electronic Designs," in *Proc. IEEE Great Lakes Symposium on VLSI*, 1998, pp. 344–350.

[84] D. Stroobandt, P. Verplaetse, and J. V. Campenhout, "Generating Synthetic Benchmark Circuits for Evaluating CAD Tools," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19(9), 2000, pp. 1011–1022.

[85] P. R. Suaris and G. Kedem, "A Quadrisection-Based Combined Place and Route Scheme for Standard Cells," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1989, vol. 8(3), pp. 234–244.

[86] W.-J. Sun and C. Sechen, "Efficient and Effective Placement for Very Large Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14(5), 1995, pp. 349–359.

[87] K. Takahashi, K. Nakajima, M. Terai, and K. Sato, "Min-Cut Placement With Global Objective Functions for Large Scale Sea-Of-Gates Arrays," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14(4), 1995, pp. 434–446.

[88] R. S. Tsay, E. S. Kuh, and C. P. Hsu, "PROUD: A Sea-of-Gates Placement Algorithm," *IEEE Design & Test of Computers*, 1988, pp. 44–56.

[89] V. V. Vazirani, *Approximation Algorithms*, First ed. Springer, 2001.

[90] K. Vorwerk, A. Kennings, and A. Vannelli, "Engineering Details of a Stable Force-Directed Placer," in *Proc. IEEE International Conference on Computer Aided Design*, 2004, pp. 573–580.

[91] J. Vygen, "Algorithms for Large-Scale Flat Placement," in *Proc. ACM/IEEE Design Automation Conference*, 1997, pp. 746–751.

[92] M. Wang, X. Yang, and M. Sarrafzadeh, "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits," in *Proc. IEEE International Conference on Computer Aided Design*, 2001, pp. 260–263.

[93] Q. Wang, D. Jariwala, and J. Lillis, "A Study of Tighter Lower Bounds in LP Relaxation Based Placement," in *Proc. IEEE Great Lakes Symposium on VLSI*, 2005, pp. 498–502.

[94] J. Warrington, R. Todd, and D. W. (Eds.), *Microarrays and Cancer Research*. BioTechniques Press/Eaton Pub., Westboro, MA, 2002.

[95] B. X. Weis and D. A. Mlynski, "A Graph Theoretical Approach to the Relative Placement Problem," *IEEE Trans. on Circuits and Systems*, vol. 35(3), 1988, pp. 286–293.

[96] D. Wismer and R. Chattergy, *Introduction to Nonlinear Optimization: A Problem Solving Approach*, First ed. Elsevier, 1978.

[97] X. Yan, B. K. Choi, and M. Sarrafzadeh, "Routability Driven White Space Allocation for Fixed-Die Standard-Cell Placement," in *Proc. ACM/IEEE International Symposium on Physical Design*, 2002, pp. 42–47.

[98] M. Yildiz and P. Madden, "Global Objectives for Standard-Cell Placement," in *Proc. IEEE Great Lakes Symposium on VLSI*, 2001, pp. 68–72.

[99] M. C. Yildiz and P. H. Madden, "Improved Cut Sequences for Partitioning Based Placement," in *Proc. ACM/IEEE Design Automation Conference*, 2001, pp. 776–779.

[100] K. Zhong and S. Dutt, "Effective Partition-Driven Placement With Simultaneous Level Processing and Global Net Views," in *Proc. IEEE International Conference on Computer Aided Design*, 2000, pp. 171–176.