Approximation Algorithms for VLSI Routing

A Thesis
Presented to
The Academic Faculty

by

Ion I. Măndoiu

In Partial Fulfillment
of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Georgia Institute of Technology
August 2000

Approximation Algorithms for VLSI Routing

Approved:

Vijay V. Vazirani, Chairman

Kalomire-Eleni Mihail

Dana Randall

H. Venkateswaran

Alexander Zelikovsky

Date Approved

Preface

This thesis gives improved approximation algorithms and heuristics for several NP-hard problems arising in the global routing phase of physical VLSI design. In each of these problems interconnection topologies must be specified for nets consisting of a source and multiple sink terminals. Different optimization objectives are used, depending on the functionality of the nets. We address the single-net routing problem under three of the most important objectives: minimizing length, skew, and number of buffers. We also address a multi-net global buffered routing problem in which a large number of nets must be routed simultaneously using only buffers located in a given set of regions, each with prescribed capacity.

The problem of finding a minimum-length interconnection of a net using only horizontal and vertical wires, the so called *rectilinear Steiner tree* (RST) problem, has long been one of the fundamental problems in the field of electronic design automation. In this thesis we give a new RST heuristic which has at its core a recent 3/2 approximation algorithm of Rajagopalan and Vazirani for the metric Steiner tree problem on quasi-bipartite graphs—these are graphs that do not contain edges connecting pairs of Steiner vertices. Our new RST heuristic achieves an excellent running time by combining an efficient implementation of the RV algorithm with simple, but powerful geometric reductions. Experiments conducted on both random and real VLSI instances show that the new RST heuristic runs significantly faster than the best existing RST heuristics and exact algorithms. Moreover, the new heuristic typically gives higher-quality solutions than previously best heuristics.

The clock skew is the maximum difference in arrival times of the clock signal at synchronizing elements. Obtaining zero- or bounded-skew clock routing is critical for maximizing the clock rate of today's deep-submicron VLSI designs. At the same time, due

to power consumption, signal integrity, and area utilization considerations, it is necessary to minimize the total wirelength used by the clock tree. The problems of finding zero- and bounded-skew clock trees with minimum total wirelength have received much attention in the VLSI CAD literature. However, the first strongly polynomial algorithms with proven constant approximation factors have been proposed only recently: Charikar et al. [16] have given $2e \approx 5.44$ and 16.86-approximation algorithms for zero- and bounded-skew trees, respectively. In this thesis we give practical algorithms with improved approximation factors for both problems. For n points in the rectilinear plane, our algorithms find, in $O(n \log n)$ time, zero- and bounded-skew trees of length at most 3 and 9 times the optimum. In general metric spaces, the respective approximation factors are 4 and 14, and can be guaranteed in $O(n^2)$ time.

As integrated circuit technology scales into the deep-submicron range, the effect of interconnect on chip performance becomes increasingly dominant. An important step in maintaining reasonable signal delay is to ensure that the length of each wire segment is whithin prescribed bounds; this can be achieved by buffer insertion. Since buffers occupy a significant area on the chip and introduce additional power requirements, the goal of buffered routing is to meet the wire segment upper-bound using the minimum number of buffers. In this thesis we consider two problems related to buffered routing. The first problem is to find a routing with minimum number of buffers for a single net, subject to upper-bounds on the length of each wire segment. We give a tight analysis of the MST heuristic recently introduced by G.-H. Lin and G. Xue for this problem. The approximation factor of the heuristic is shown to be one less than the MST number of the underlying space, defined as the maximum possible degree of a minimum-degree MST spanning points from the space. In particular, on instances drawn from the rectilinear plane, the MST heuristic has a tight approximation factor of 3.

The second buffered routing problem addressed in this thesis is how to perform simultaneous buffering of a large number of nets, given an existing buffer block plan, i.e., using buffers located at a given set of buffer blocks, each with limited capacity. We give a provably good algorithm based on a novel approach to multiterminal multicommodity flow approximation inspired by recent results of Garg and Könemann [35] and Fleischer [29]. Our algorithm routes the nets subject to both upper and lower bounds on the length of wire segments, as well as path-length upper bounds and buffer parity constraints per connection. The new algorithm outperforms existing algorithms for the problem [20], and has been validated on top-level layouts extracted from a recent high-end microprocessor design.

Contents

List of Algorithms

List of Figures

List of Tables

Acknowledgements

First and foremost, I would like to thank my advisor, Vijay Vazirani, for his continuous guidance, advice, and support throughout my years at Georgia Tech. Vijay initiated me to approximation algorithms, and encouraged my efforts to combine theoretical work in this field with experimental validation.

I am also grateful to Alex Zelikovsky, who has timely accepted a position with the Computer Science department of Georgia State University just as I was trying to learn more about VLSI CAD. He ended up acting as my second advisor for the past year and a half, and I have greatly benefited from our close collaboration.

The results in this thesis have been obtained in joint work with Feodor Dragan, Joe Ganley, Andrew Kahng, Sudhakar Muddu, Vijay Vazirani, and Alex Zelikovsky. Working with them has been a rewarding experience. I wish to thank my coauthors for giving me this opportunity and for allowing me to include our joint results in this thesis.

During my Ph.D. studies at Georgia Tech I benefited in many ways from interactions with faculty members of the College of Computing, and of the Mathematics and Industrial Systems Engineering departments. Extra thanks go to Howard Karloff, Milena Mihail, Dana Randall, Leonard Schulman, Prasad Tetali, and H. Venkateswaran for serving on my Ph.D. examination, proposal, and/or defense committees. I would also like to express my deepest gratitude to Cristian Calude and Ioan Tomescu for guiding my first research steps at the University of Bucharest.

Last, but not least, I would like to thank my family—my faraway parents and brother, my wife and our daughter—for their love and support. This thesis is dedicated to them.

Chapter 1

Introduction

Physical VLSI design is the process of translating the electrical description of a circuit into a geometrical layout. Obtaining good solutions to the NP-hard problems arising in this process is crucial for the production of low-cost, high-performance integrated circuits. In this thesis we explore, both theoretically and experimentally, several approximation algorithms and heuristics for problems related to the *global routing* phase of physical design.

In global routing, interconnection topologies must be specified for a large number of *signal nets*, each consisting of a *source* and multiple *sink* terminals. Routing is typically performed one net at a time, a feasible solution to a single-net instance of the routing problem being a rectilinear Steiner tree for the set of terminals. Different optimization objectives are used, depending on the functionality of the nets. The most important objectives considered in the VLSI literature are [50]:

- **Length.** Minimizing length has long been the prevailing objective in VLSI routing, since a minimum-length interconnection occupies the minimum amount of area and has minimum overall capacitance and resistance. Although recent advances of integrated circuit technology into the deep-submicron realm have introduced additional routing objective functions, minimizing length remains the most important objective for non-critical nets and in physically small instances. This objective is captured by the the well-studied *Rectilinear Steiner Tree* (RST) problem.

- **Delay.** As VLSI technology scales to smaller feature sizes and larger layout areas, propagation delay increasingly dominates delay through switching devices. In performance-driven routing, one seeks to control the propagation delay between the source and a specified set of sinks. A standard formulation of this objective is the *Rectilinear Steiner Arborescence* (RSA) problem, that asks for a minimum length "shortest-path" rectilinear Steiner tree rooted at the source, i.e., a minimum length Steiner tree in which the length of each source-to-sink path is as small as possible.

- **Skew.** In order to maximize the clock rate it is necessary to minimize the *skew* of the clock network, i.e., the maximum difference between source-to-sink delays. Two formulations capturing this objective have received much attention in the VLSI literature. The *Zero-Skew Tree* (ZST) problem is to find a minimum length rooted rectilinear Steiner tree in which all root-to-leaf paths have equal length. The *Bounded-Skew Tree* (BST) problem is defined similarly, except that the length of two root-to-leaf paths may differ by at most a given number b.

- **Buffers.** Buffer insertion is an increasingly popular solution for maintaining signal integrity and achieving reasonable signal delay in the global nets of today's interconnect-dominated deep-submicron designs. To minimize the increase in area and power requirements, it is desirable to use the smallest number of buffers that meets the given upper-bound on buffer–buffer and buffer–terminal wire lengths. The *Minimum number of Steiner Points Tree* (MSPT) problem captures this objective by modeling buffers as Steiner points, possibly of degree 2.

In Chapters 2–4 of this thesis we consider the single-net routing problem under three of the above objectives: minimizing length, skew, and number of buffers. In the final chapter we address a multi-net global buffered routing problem in which a large number of nets must be routed simultaneously using only buffers located at a given set of buffer blocks, each with limited capacity. In the remaining of this chapter we formally introduce these problems and give a summary of our results.

1.1 Rectilinear Steiner trees

Since VLSI fabrication technology typically mandates the use of only horizontal and vertical interconnect, the problem that captures the length-minimization objective in global VLSI routing is the following variant of the classical Steiner tree problem:

Rectilinear Steiner Tree Problem: Given a set of terminals in the plane, find a minimum length interconnection of the terminals, using only horizontal and vertical wires. Wires are allowed to meet at points other than the terminals, these non-terminal meeting points are referred to as *Steiner points*.

The RST problem was defined by Hanan in 1966 [40], and has been the subject of active research ever since. Since the RST problem is NP-hard [34], most of the research effort on the problem has been devoted to designing heuristics and approximation algorithms, see e.g. [1, 8, 11, 31, 36, 41, 45, 50, 53, 61, 75, 77]. In an extensive survey of RST heuristics up to 1992 [46], the Batched Iterated 1-Steiner (BI1S) heuristic of Kahng and Robins [49] emerged as the clear winner, with an average improvement over the MST on terminals of almost 11%.

After a steady, but relatively slow progress [30, 33, 68], exact RST algorithms have recently witnessed spectacular progress [73], with the new release of the GeoSteiner code by Warme, Winter, and Zachariasen matching in average running time the fast BI1S implementation of Robins. We are thus faced with the paradoxical situation that an exact algorithm for an NP-hard problem has the same average running time as a state-of-the-art heuristic for the problem.

In Chapter 2 of this thesis we give a new RST heuristic that improves over the BI1S heuristic of Kahng and Robins in both speed (the new heuristic is faster by a factor of 2–10, depending on the instance size) and average solution quality. The new heuristic comes within 0.5% of the optimum solution computed by GeoSteiner on the average, and runs 4–10 times faster than the exact code.

Our results are obtained by exploiting a number of recent algorithmic and implementation ideas. On the algorithmic side, we build on the recent $3/2$ approximation algorithm of Rajagopalan and Vazirani [64] for the metric Steiner tree problem on quasi-bipartite graphs; these are graphs that do not contain edges connecting pairs of Steiner vertices. This algorithm is based on the linear programming relaxation of a sophisticated integer formulation of the metric Steiner tree problem, called the bidirected cut formulation. It is well known that the RST problem can be reduced to the metric Steiner tree problem on graphs [40], however, the graphs obtained from the reduction are not quasi-bipartite. We give an RV-based heuristic for finding Steiner trees in arbitrary (non quasi-bipartite) metric graphs. The heuristic, called *Iterated RV* (IRV), computes a Steiner tree of a quasi-bipartite subgraph of the original graph using the RV algorithm, in order to select a set of candidate Steiner vertices. The process is repeated with the selected Steiner vertices treated as terminals—thereby allowing the algorithm to pick larger quasi-bipartite subgraphs, and seek additional Steiner vertices for inclusion in the tree—until no further improvement is possible.

The efficient implementation of the IRV heuristic depends critically on the size of the quasi-bipartite subgraphs considered in each iteration. We decrease the size of the graphs that correspond to RST instances by applying *reductions*, which are deletions of edges and vertices that do not affect the quality of the result. Our key edge reduction is based on Robins and Salowe's result that bounds the maximum degree of a rectilinear MST [66], and allows us to retain in the graph at most 4 edges incident to each vertex. Notably, the same reduction is the basis of a significant speed-up in the running time of BI1S [37], and is currently incorporated in Robins' implementation. Our vertex reduction is based on a simple *empty rectangle* test [8, 31, 76].

It is interesting to note that, due to poor performance and prohibitive running times, none of the previous algorithms with proven guarantees for the Steiner tree problem in graphs [1, 8, 36, 61, 77] was found suitable as the core algorithmic idea around which heuristics can be built for use in the industry. Our adaptation of the RV algorithm fills this void for the first time, and points to the importance of drawing on the powerful new ideas developed recently in the emerging area of approximation algorithms for NP-hard optimization problems.

1.2 Zero- and bounded-skew clock trees

Today's high-performance VLSI circuits use almost exclusively synchronous designs. In these circuits, a clock signal, distributed by means of a tree rooted at the clock source, must be delivered periodically to a set of clock sinks. To achieve maximum clock rate it is necessary to minimize the *clock skew*, i.e., the maximum difference in arrival times of the clock signal at synchronizing elements.

Clock skew can be controlled in a number of ways, e.g., by using wires with non-uniform width or by inserting buffers. In this thesis we address the most popular approach of controlling skew, which is to control the length of wires in the clock tree. In this approach, a feasible routing for a set of sinks S is a *zero-skew tree* (ZST), i.e., a rooted Steiner tree in which all root-to-sink paths have equal length. Due to power consumption, signal integrity, and area utilization considerations, the objective is to minimize the total length of the ZST. Thus the clock tree construction problem has been formalized [5] as follows:

Rectilinear Zero-Skew Tree Problem: Given a set S of sinks in the rectilinear plane, find a zero-skew tree of minimum total length for S.

As noted in [50], a more realistic design requirement is captured by bounded-skew trees (BST). A rooted Steiner tree T for the set S of sinks is a *b-bounded-skew tree* if the difference in length between any two root-to-sink paths is at most b.

Rectilinear Bounded-Skew Tree Problem: Given a set S of sinks in the plane and bound $b > 0$, find a b-bounded-skew tree of minimum total length for S.

The rectilinear BST problem and the generalization of the ZST problem to arbitrary metric spaces are NP-hard [16]. The complexity of the rectilinear ZST problem is not known—for a fixed tree topology the problem can be solved in linear time by using the *Deferred-Merge Embedding* (DME) algorithm independently introduced in [10, 13, 24].

Although the rectilinear zero- and bounded-skew tree problems have received much attention in the VLSI CAD literature [6, 10, 13, 14, 19, 24, 25, 47, 18, 54] (see Chapter 4 of [50] for a detailed review), the first algorithms with constant approximation factors have been proposed only recently, by Charikar et al. [16]. Charikar et al. generalize the ZST and BST problems to arbitrary metric spaces, and, for this general setting, give algorithms with approximation factors of $2e \approx 5.44$ and 16.86, respectively. The BST algorithm in [16] relies on an approximation algorithm for the Steiner tree problem in graphs. Using the currently best Steiner tree approximation of Robins and Zelikovsky [67] and Arora's PTAS for computing rectilinear Steiner trees [3, 4], the BST bounds in [16] can be updated to 16.11 for arbitrary metric spaces, and to 12.53 for the rectilinear plane.

In Chapter 3 of this thesis we give practical algorithms with improved approximation factors for both problems. For n points in the rectilinear plane, our algorithms find zero- and bounded-skew trees of length at most 3 and 9 times the optimum. In general metric spaces, the respective approximation factors are 4 and 14.

An important feature of our algorithms is their practical running time: our algorithms run in $O(n \log n)$ time for the rectilinear plane and in $O(n^2)$ time for arbitrary metric spaces. Thus, our algorithms can easily handle the clock nets with hundreds of thousands of sinks that occur in large cell-based or multi-chip module designs.

1.3 Bounded edge-length Steiner trees with minimum number of Steiner points

Bounded edge-length Steiner trees are a natural model for applications arising in VLSI routing as well as wireless network design. In these applications terminals are points in the plane, and the underlying metric is either the rectilinear metric, L_1, as in buffer insertion for clock delay and skew minimization, or the Euclidean metric, L_2, as in the design of fixed wireless networks. The goal is to minimize the number of Steiner points, which correspond to buffers, respectively radio relays.

Minimum Number of Steiner Points Tree Problem: Given a set S of terminals in an arbitrary metric space and bound $R > 0$, find a Steiner tree for S with minimum number of Steiner points among the trees with edges of length at most R.

The MSPT problem, which is a special case of the *node-weighted* Steiner tree problem [51], was first introduced by Sarrafzadeh and Wong [69]. The problem is NP-hard even when restricted to points in the rectilinear or Euclidean planes [69]. The results of [51] and [28] imply that, for arbitrary metric spaces, the MSPT problem cannot be approximated within a factor of $(1 - \varepsilon)\ln n$, where n is the number of terminals, unless NP $\subseteq$ TIME$(n^{\log\log n})$. Thus, the $\ln n$-approximation algorithm of Guha and Kuller [38] is optimal in this case.

Optimal approximation results are not known for the rectilinear and Euclidean planes. Recently, Lin and Xue [55] considered the following *MST heuristic* for the MSPT problem: Compute an MST on terminals, then subdivide each edge (u, v) of the MST via $\lceil d(u, v)/R \rceil - 1$ equally spaced Steiner points, where $d(u, v)$ stands for the distance between u and v. Lin and Xue proved that the MST heuristic has an approximation factor not worse than 5 in the Euclidean plane, leaving open the problem of finding the exact approximation factor.

In Chapter 4 of this thesis we give a tight analysis of the MST heuristic for any L_p metric space, showing that its approximation factor is exactly one less than the *MST* *number*, defined as the maximum possible degree of a minimum-degree MST spanning points from the space. Since the MST numbers for the rectilinear and Euclidean planes are 4 and 5 [66], our analysis implies that for these two metric spaces the MST heuristic has tight approximation factors of 3 and 4, respectively.

1.4 Multi-net global routing via buffer blocks

Process scaling leads to an increasingly dominant effect of interconnect on high-end chip performance. Each top-level global net must undergo repeater insertion to maintain signal integrity and reasonable signal delay. Estimates of the need for repeater insertion range up to 10^6 repeaters for top-level on-chip interconnect for 50nm technology. These repeaters occupy a significant area on the chip, affect global routing congestion, can entail non-standard cell height and special power routing requirements, and can act as noise sources. In a block- or reuse-based methodology, designers seek to isolate repeaters for global interconnect from individual block implementations.

For these reasons, a *buffer block* methodology has become increasingly popular in structured-custom and block-based ASIC methodologies. In Chapter 5 of this thesis we address the problem of how to perform buffering of global nets *given an existing buffer block plan*. We give a provably good algorithm based on a recent approach of Garg and Könemann [35] and Fleischer [29]. Our method routes the nets using available buffer blocks subject to both upper and lower bounds on repeater intervals, as well as path-length upper bounds and buffer parity constraints per connection. More formally, our problem is defined as follows.

Given:

- a planar region with rectangular obstacles;
- a set of nets in the region, each net has:

    - a single source and one or more sinks;
    - a non-negative importance (criticality) coefficient;

- each sink has:
    - a parity requirement, which specifies the required parity of the number of buffers (inverters) on the path connecting it to the source;
    - a timing-driven requirement, which specifies the maximum number of buffers allowed on this path;
- a set of buffer blocks, each with given capacity; and
- an interval $[L, U]$ specifying lower and upper bounds on the distance between buffers.

The **Global Routing via Buffer Blocks (GRBB) Problem** is to route a subset of the given nets, with maximum total importance, such that:

- the distance between the source of a route and its first repeater, between any two consecutive repeaters, respectively between the last repeater on a route and the route's sink, are all between L and U;
- the number of trees passing through any given buffer block does not exceed the block's capacity;
- the number of buffers on each source-sink path should not exceed the given upper bound and should be of the given parity; to meet the parity constraint two buffers of the same block can be used.

If possible, the optimum solution to the GRBB problem simultaneously routes all the nets. Otherwise, it maximizes the sum of the importance coefficients over routed nets. The importance coefficients can be used to model various practical objectives. For example, importance coefficients of 1 for each net correspond to maximizing the number of routed nets, and importance coefficients equal the number of sinks in the net correspond to maximizing the number of connected sinks.

In Chapter 5 of this thesis we show that the GRBB problem can be formulated as a generalized version of (vertex-capacitated) integer multiterminal multicommodity flow (MTMCF). Exploiting this formulation, we give a new algorithm for the GRBB problem based on randomized rounding of an approximate solution to the fractional relaxation of the integer MTMCF program. Prior to our work, multicommodity flow based heuristics have been applied [60, 70, 12, 43, 2] to unbuffered versions of VLSI global routing in which the main constraints are given by *edge*, not *vertex*, capacities. As noted in [56], the applicability of these algorithms has often been limited to problem instances of relatively small size by the prohibitive cost of solving exactly the fractional relaxation. Following [2], we avoid this limitation by using an *approximate* MTMCF algorithm. This algorithm, based on recent results of [35, 29], allows for a smooth trade-off between running time and solution accuracy. Our experiments show that even MTMCF solutions with low accuracy give good final solutions for the GRBB problem.

An interesting feature of our algorithm is its ability to work with multiterminal nets—previous work on the GRBB problem [20, 71] has considered only the case of 2-pin nets. Experiments on top-level layouts extracted from a recent high-end microprocessor design validate our MTMCF-based algorithm, and indicate that (1) the algorithm significantly outperforms existing algorithms for the problem [20], even when applied to 2-pin net decompositions, and (2) applying the MTMCF algorithm on multipin nets instead of 2-pin decompositions further increases the quality of the solution, even when the same time budget is given to both algorithms.

Chapter 2

A new heuristic for rectilinear Steiner trees*

2.1 Introduction

The rectilinear Steiner tree (RST) problem is that of finding a minimum-length interconnection of a set of terminals in the plane using only horizontal and vertical wires. The RST problem was introduced by Hanan in 1966 [40], and has been the subject of active research ever since, mostly because of its aplications in electronic design automation. Although recent advances of integrated circuit technology into the deep-submicron realm have introduced additional routing objectives besides length minimization, the Steiner tree problem retains its importance for non-critical nets and in physically small instances.

Since the RST problem is NP-hard [34], most of the research effort on the problem has been devoted to designing heuristics and approximation algorithms, see e.g. [1, 8, 11, 31, 36, 41, 45, 49, 53, 61, 75, 77]. In an extensive survey of RST heuristics up to 1992 [46], the Batched Iterated 1-Steiner (BI1S) heuristic of Kahng and Robins [49] emerged as the clear winner with an average improvement over the MST on terminals of almost 11%. Subsequently, two other heuristics [11, 53] have been reported to match the same performance.

After a steady, but relatively slow progress [30, 33, 68], exact RST algorithms have recently witnessed spectacular progress [73] (see also [32]). The new release [74] of the GeoSteiner code by Warme, Winter, and Zachariasen has average running time comparable to the fast BI1S implementation of Robins [65] on random instances. We are thus faced

*This chapter is based on joint work with Vijay V. Vazirani and Joseph L. Ganley [57, 58].

with the paradoxical situation that an exact algorithm for an NP-hard problem has the same average running time as a state-of-the-art heuristic for the problem. It appears that, for the RST problem, progress on heuristics has lagged behind that on exact algorithms.

We try to remedy this situation by proposing a new RST heuristic. Our experiments show that the new heuristic has better average running time than both Robins' implementation of BI1S and the GeoSteiner code. Moreover, the new heuristic gives higher-quality solutions than BI1S on the average; of course, it cannot beat GeoSteiner in solution quality.

Our results are obtained by exploiting a number of recent algorithmic and implementation ideas. On the algorithmic side, we build on the recent $3/2$ approximation algorithm of Rajagopalan and Vazirani [64] for the metric Steiner tree problem on quasi-bipartite graphs; these are graphs that do not contain edges connecting pairs of Steiner vertices. This algorithm is based on the linear programming relaxation of a sophisticated integer formulation of the metric Steiner tree problem, called the bidirected cut formulation. It is well known that the RST problem can be reduced to the metric Steiner tree problem on graphs [40], however, the graphs obtained from the reduction are not quasi-bipartite. We give an RV-based heuristic for finding Steiner trees in arbitrary (non quasi-bipartite) metric graphs. The heuristic, called *Iterated RV* (IRV), computes a Steiner tree of a quasi-bipartite subgraph of the original graph using the RV algorithm, in order to select a set of candidate Steiner vertices. The process is repeated with the selected Steiner vertices treated as terminals—thereby allowing the algorithm to pick larger quasi-bipartite subgraphs, and seek additional Steiner vertices for inclusion in the tree—until no further improvement is possible.

The efficient implementation of the IRV heuristic depends critically on the size of the quasi-bipartite subgraphs considered in each iteration. We decrease the size of the graphs that correspond to RST instances by applying *reductions,* which are deletions of edges and vertices that do not affect the quality of the result. Our key edge reduction is based on Robins and Salowe's result that bounds the maximum degree of a rectilinear MST [66], and allows us to retain in the graph at most 4 edges incident to each vertex. Notably, the same reduction is the basis of a significant speed-up in the running time of BI1S [37], and is currently incorporated in Robins' implementation [65]. Our vertex reduction is based on a simple *empty rectangle* test that has its roots in the work of Berman and Ramaiyer [8] (see also [31, 76]).

We ran experiments to compare our implementation of IRV against Robins' implementation of BI1S [65] and against the GeoSteiner code of Warme, Winter, and Zachariasen [74]. The results reported in Section 2.4 show that, on both random and real VLSI instances, our new heuristic produces on the average higher-quality solutions than BI1S. The quality improvement is not spectacular, but we should note that solutions produced by BI1S are already less than 0.5% away from optimum on the average.

More importantly, IRV's improvement in solution quality is achieved with an excellent running time. On random instances with up to 250 terminals, our IRV code runs 4–10 times faster than the lp_solve based version of GeoSteiner used in our experiments, and 2–10 times faster than Robins' implementation of BI1S—the speed-up increases with the number of terminals. After noticing that BI1S can also benefit from vertex reductions, we incorporated the empty rectangle test into Robins' BI1S code. The enhanced BI1S code becomes about 30% faster than our IRV code on large random instances. However, this does not necessarily mean that BI1S is the best heuristic in practice. Results on real VLSI instances indicate a different hierarchy: On these instances both IRV and GeoSteiner are faster than the enhanced BI1S.

It is interesting to note that, due to poor performance and prohibitive running times, none of the previous algorithms with proven guarantees for the Steiner tree problem in graphs [1, 8, 36, 61, 77] was found suitable as the core algorithmic idea around which heuristics can be built for use in the industry. Our adaptation of the RV algorithm fills this void for the first time, and points to the importance of drawing on the powerful new ideas developed recently in the emerging area of approximation algorithms for NP-hard optimization problems [72].

The remainder of this chapter is structured as follows. Section 2.2 describes the RV algorithm and its extension to non quasi-bipartite graphs. Section 2.3 describes how this extension, IRV, is used to solve RST instances, and Section 2.4 presents experimental results comparing IRV with BI1S and GeoSteiner on test cases both randomly generated and extracted from real circuit designs.

2.2 Steiner trees in graphs

The *metric Steiner tree in graphs* (GST) problem is: Given a connected graph $G = (V, E)$ whose vertices are partitioned in two sets, T and S, the *terminal* and *Steiner* vertices respectively, and non-negative edge costs satisfying the triangle inequality, find a minimum cost tree spanning all terminals and any subset of the Steiner vertices. Recently, Rajagopalan and Vazirani [64] presented a 3/2 approximation algorithm (henceforth refered to as the RV algorithm) for the GST problem when restricted to quasi-bipartite graphs, i.e., graphs that have no edge connecting a pair of Steiner vertices. In this section we review the RV algorithm, discuss its implementation, and present an RV-based heuristic for the GST problem on arbitrary graphs.

2.2.1 The bidirected cut relaxation

The RV algorithm is based on a sophisticated integer programming (IP) formulation of the GST problem. A related, but simpler formulation is given by the following observation: A set of edges $E' \subseteq E$ connects terminals in T if and only if every cut of G separating two terminals crosses at least one edge of E'. The IP formulation resulting from this observation is called the undirected cut formulation. The IP formulation on which the RV algorithm is based, called the *bidirected cut formulation*, is obtained by considering a directed version of the above cut condition.

Let $\vec{E}$ be the set of arcs obtained by replacing each undirected edge $(u, v) \in E$ by two directed arcs $u \to v$ and $v \to u$. For a set C of vertices, let $\delta(C)$ be the set of arcs $u \to v$ with $u \in C$ and $v \in V \setminus C$. Finally, if t_o is a fixed terminal, let $\mathcal{C}$ contain all sets $C \subseteq V$ that contain at least one terminal but do not contain t_o. The bidirected cut formulation attempts to pick a minimum cost collection of arcs from $\vec{E}$ in such a way that each set in $\mathcal{C}$ has at least one outgoing arc:

$$\begin{array}{lll} \text{minimize} & \sum_{e \in \vec{E}} cost(e) x_e & \\ \text{s.t.} & \sum_{e:\, e \in \delta(C)} x_e \geq 1, & C \in \mathcal{C} \\ & x_e \in \{0, 1\}, & e \in \vec{E} \end{array} \tag{1}$$

By allowing x_e's to assume non-negative fractional values we obtain a linear program (LP) called the *bidirected cut relaxation* of the GST problem:

$$\begin{array}{lll} \text{minimize} & \sum_{e \in \vec{E}} cost(e) x_e & \\ \text{s.t.} & \sum_{e:\, e \in \delta(C)} x_e \geq 1, & C \in \mathcal{C} \\ & x_e \geq 0, & e \in \vec{E} \end{array} \tag{2}$$

The *dual* of the covering LP (2) is the packing LP:

$$\begin{array}{lll} \text{maximize} & \sum_{C \in \mathcal{C}} y_C & \\ \text{s.t.} & \sum_{C:\, e \in \delta(C)} y_C \leq cost(e), & e \in \vec{E} \\ & y_C \geq 0, & C \in \mathcal{C} \end{array} \tag{3}$$

From LP-duality theory, the cost of any feasible solution to (3) is less than or equal to the cost of the optimum solution to (2), and hence, less than or equal to the cost of any feasible solution to (1). The RV algorithm uses this observation to guarantee the quality of the solution produced: The algorithm constructs feasible solutions to both IP (1) and LP (3), in such a way that the costs of the two solutions differ by at most a factor of 3/2.

2.2.2 The RV algorithm

The RV algorithm works on quasi-bipartite graphs G. At a coarse level, the RV algorithm is similar to the Batched Iterated 1-Steiner algorithm of Kahng and Robins [49]: Both algorithms work in phases, and in each phase some Steiner vertices are iteratively added to the set of terminals. While BI1S adds Steiner vertices to T greedily—based on the decrease in the cost of the MST—the RV algorithm uses the bidirected cut relaxation to guide the addition.

In each phase, the RV algorithm constructs feasible solutions to both IP (1) and LP (3). The bidirected cut formulation and its relaxation are inherently asymmetric, since they require a terminal t_o to be singled out. However, the RV-Phase algorithm works in a symmetric manner: The information it computes can be used to derive feasible solutions for any choice of t_o.

A set $C \subseteq V$ is called *proper* if both C and $V \setminus C$ contain terminals; with respect to the original set of terminals only sets in $\mathcal{C}$ and their complements are proper. During its execution, the RV-Phase algorithm tentatively converts some Steiner vertices into terminals; note that the only proper sets created by these conversions are singleton sets containing new terminals, and their complements. The algorithm maintains a variable y_C, called *dual*, for every proper set, including the newly created ones. The *amount of dual felt* by arc e is $\sum_{C\,:\,e \in \delta(C)} y_C$; we say that e is *tight* when $\sum_{C\,:\,e \in \delta(C)} y_C = cost(e)$. A set C of vertices is *unsatisfied* if it is proper and $\delta(C)$ does not contain any tight arc.

The RV-Phase algorithm (Algorithm 1) starts with y_C set to 0 for every proper set C, and an empty list $\vec{L}$ of tight arcs. It then proceeds in a *primal-dual* manner, by alternatively

```
Input: Bidirected quasi-bipartite graph $G = (V, \vec{E})$, set $T \subseteq V$ of terminals
Output: Augmented set $T$

1. $\vec{L} \leftarrow \emptyset$; For each proper set $C$, $y_C \leftarrow 0$
2. If all proper sets are satisfied by arcs in $\vec{L}$, return $T$ and exit
3. Otherwise, uniformly raise the $y$ values of minimally unsatisfied sets until an arc
   $u \rightarrow v$ goes tight
4. If $u \notin T$, then $T \leftarrow T \cup \{u\}$; repeat from Step 1
5. Else, $\vec{L} \leftarrow \vec{L} \cup \{u \rightarrow v\}$; repeat from Step 2
```

Algorithm 1: The RV-Phase algorithm

raising dual variables as long as this does not violate the packing constraints of (3), and picking tight edges into $\vec{L}$, thus satisfying more and more proper sets. When the algorithm stops, all proper sets are satisfied by tight arcs in $\vec{L}$.

Theorem 1 *[64] (a) If arc $u \rightarrow v$, $u \notin T$, goes tight then* $cost(\text{MST}(T \cup \{u\})) < cost(\text{MST}(T))$.
(b) At the end of the RV-Phase *algorithm,* $cost(\text{MST}(T \cup \{u\})) \geq cost(\text{MST}(T))$ *for every $u \notin T$.*

The RV algorithm (whose pseudo-code we omit) repeats the RV-Phase algorithm followed by removal of unnecessary Steiner vertices, until no further improvement is made in the cost of MST(T). At the end of the algorithm, the duals raised around proper sets are converted into a solution to (1) by picking t_o and discarding y_S's with $S \notin \mathcal{C}$. The 3/2 approximation guarantee follows by relating the cost of this solution to the cost of MST(T).[1]

[1]The tree produced by the RV algorithm is *locally optimal*, i.e., cannot be improved by the adition or deletion of a single Steiner vertex. Recently, Robins and Zelikovsky [67] used a different argument to prove that *any* locally optimal Steiner tree gives a 3/2-approximation for the metric Steiner tree in quasi-bipartite graphs.

2.2.3 Efficient implementation of the RV-Phase algorithm

Since our heuristic for general graphs uses the RV-Phase algorithm as a subroutine, we describe here an efficient implementation of it. Several implementation ideas are derived from the following key property maintained throughout the RV-Phase algorithm:

Lemma 2 *[64] Let u and v be two terminals. If all arcs along some path $u \rightarrow x_1 \rightarrow \cdots \rightarrow x_k \rightarrow v$ are tight, then so are the arcs on the reverse path, $v \rightarrow x_k \rightarrow \cdots \rightarrow x_1 \rightarrow u$.*

For implementation purposes we do not need to keep track of the duals raised; all that matters is the order in which arcs get tight. The tightening time of an arc can be determined by monitoring the number of minimally unsatisfied sets (henceforth called *active* sets) that are felt by that arc.

It is easy to see that the set of vertices reachable via tight arcs from a terminal u forms an active set; Lemma 2 implies that no other active set can contain u. Thus, we get:

Corollary 3 *For any terminal u, there is exactly one active set containing u at any time during the algorithm. Hence, the tightening time of any arc $u \rightarrow v$, $u \in T$, is exactly $cost(u, v)$.*

Unlike terminals, Steiner vertices may be contained in multiple active sets. Hence, arcs out of Steiner vertices will feel dual at varying rates during the algorithm.

Lemma 4 *Let u be a Steiner vertex. If arc $u \rightarrow v$ goes tight in the RV-Phase algorithm, then arc $v \rightarrow u$ goes tight at the same time or before $u \rightarrow v$ does. Moreover, each arc $u \rightarrow w$ for which $w \rightarrow u$ is already tight will go tight together with $u \rightarrow v$.*

Proof : In order to get tight, $u \rightarrow v$ must feel some active set, i.e., there must exist a tight path from a terminal $v' \neq v$ to u. After $u \rightarrow v$ gets tight, there is a tight path from v' to v, and, by Lemma 2, the reverse path (hence the arc $v \rightarrow u$) must also be tight. The second claim follows similarly. □

Since several arcs out of a Steiner vertex get tight simultaneously, we say that the vertex *crystallizes* when this happens. Note that crystallization is precisely the moment when the vertex is added to T, i.e., when it begins to be treated as terminal. Lemma 4 implies that, in order to detect when a Steiner vertex crystallizes, it suffices to monitor the amount of dual felt by the shortest arc out of that Steiner vertex, which we will call *critical arc*.

Our implementation of RV-Phase (Algorithm 2) is a discrete-event simulation of the continuous-time Algorithm 1. The structure of the algorithm is reminiscent of the well-known MST algorithm of Kruskal (see, e.g., [21]): Arcs out of terminals are sorted in non-decreasing order, then marked as tightened one by one (and active sets updated accordingly) until a Steiner vertex crystallizes or all terminals are connected. However, unlike Kruskal's algorithm, which visits each edge only once, the RV-Phase algorithm must restart the tree construction from scratch after each vertex crystallization. Our implementation exploits the fact that the two opposite arcs connecting a pair of terminals get tight at the same time, and handles one of these arcs implicitly. The main advantage of this implicit representation is that the list of arcs out of terminals does not have to be updated and re-sorted after vertex crystallizations.

In order to determine the crystallization times, we maintain for each terminal u its active set, $as(u)$, i.e., the set of vertices reachable from u by tight arcs. We also maintain for each Steiner vertex s the cost $c(s)$ of its critical arc and the number $na(s)$ of active sets containing s. Whenever $na(s)$ changes its value we update the amount $df(s)$ of dual felt by the critical arc of s. Notice that, if $na(s) > 1$, the critical arc of s feels only $na(s) - 1$ of the active sets, since one of the active sets contains both ends of the critical arc. Thus, if $na(s) > 1$, the estimate for the crystallization time of s is given by $ut(s) + (c(s) - df(s))/(na(s) - 1)$, where $ut(s)$ represents the time of the last update of $df(s)$. If $na(s) \leq 1$, the critical arc of s feels no dual, so s has an estimated crystallization time of ∞.

Input: Bidirected quasi-bipartite graph $G = (V, \vec{E})$, set $T \subseteq V$ of terminals
Output: Augmented set T

```
1. Let $a_1, \ldots, a_m$ be the arcs in $\{(u,v) \in \vec{E} | u,v \in T, u < v\} \cup$
   $\{(u,v) \in \vec{E} \mid u \in T, v \notin T\}$, sorted non-decreasingly by cost
2. $S \leftarrow V \setminus T$; $time \leftarrow 0$; $i \leftarrow 1$
     For each $t \in T$, $as(t) \leftarrow \{t\}$
     For each $s \in S$,
         $c(s) \leftarrow \min\{cost(s,u) \mid (s,u) \in \vec{E}\}$
         $df(s) \leftarrow 0$; $na(s) \leftarrow 0$; $ut(s) \leftarrow 0$
3. $(u,v) \leftarrow a_i$; $time \leftarrow cost(u,v)$
4. $C \leftarrow \{s \in S \mid na(s) > 1\}$
     If $C \neq \emptyset$ then
         $ct \leftarrow \min\{ut(s) + (c(s) - df(s))/(na(s) - 1) \mid s \in C\}$
         $s_0 \leftarrow \text{argmin}\{ut(s) + (c(s) - df(s))/(na(s) - 1) \mid s \in C\}$
     Else, $ct \leftarrow \infty$
5. If $time > ct$ then $T \leftarrow T \cup \{s_0\}$; repeat from step 2
6. Else, if $v \in S$ and $v \notin as(u)$ then
     If $na(v) > 1$, $df(v) \leftarrow df(v) + (na(v) - 1)(time - ut(v))$
     $as(u) \leftarrow as(u) \cup \{v\}$; $na(v) \leftarrow na(v) + 1$; $ut(v) \leftarrow time$
7. Else, if $v \in T$ and $v \notin as(u)$ then
     For each $s \in as(u) \cap as(v)$ do
         $df(s) \leftarrow df(s) + (na(s) - 1)(time - ut(s))$
         $na(s) \leftarrow na(s) - 1$; $ut(s) \leftarrow time$
     $as(u) \leftarrow as(v) \leftarrow as(u) \cup as(v)$
8. If $T \subseteq as(u)$ then return $T$ and exit
9. Else, $i \leftarrow i + 1$; repeat from Step 3
```

Algorithm 2: Implementation of the RV-Phase algorithm

```
Input: Arbitrary graph $G = (V, E)$, set $T \subseteq V$ of terminals
Output: Steiner tree on terminals
1. $T_{best} \leftarrow T_{in} \leftarrow T$
2. Remove from $G$ all edges $(u, v)$ with $u, v \notin T$, bidirect remaining edges, then
   run the RV-Phase algorithm on the resulting graph. This will add some Steiner
   vertices to $T$
3. Construct an MST on $T$, then prune from $T \setminus T_{in}$ all vertices with tree degree
   $\leq 2$
4. If $cost(\text{MST}(T)) < cost(\text{MST}(T_{best}))$ then
       $T_{best} \leftarrow T$; repeat from Step 2
5. Return $\text{MST}(T_{best})$.
```

Algorithm 3: The IRV algorithm

Maintaining active sets using an augmented disjoint-set data-structure leads to a worst case running time of $O(k \cdot |T| \cdot |S|)$, where k is the number of crystallized Steiner vertices—all other operations are performed in $O(k \cdot |E| \cdot \log |V|)$.

2.2.4 The iterated RV heuristic

A simple way of dealing with non-quasi-bipartite graphs is to remove all Steiner-Steiner edges and then run the RV algorithm. To allow Steiner-Steiner edges to come into play, we iterate this process. If a Steiner vertex is added to T during some run of the RV algorithm, for subsequent runs we extend the graph by adding *all* edges incident to it, not just those leading to terminals.

Preliminary experiments have shown that it is better—in both running time and solution quality—to extend the graph after running just one RV-Phase, not the full RV algorithm, on the quasi-bipartite graph. This gives Algorithm 3.

2.3 Rectilinear Steiner trees

The *rectilinear Steiner tree* (RST) problem is defined as follows: Given a set T of *terminals* in the plane, find a shortest interconnection of the terminals using only horizontal and vertical lines. Lines are allowed to meet at points other than the terminals; as usual, non-terminal meeting points are called *Steiner points*.

By a classical result of Hanan [40], there exists an optimal rectilinear Steiner tree that uses only Steiner points located at intersections of vertical and horizontal lines passing through terminals. Thus, finding a minimum rectilinear Steiner tree on a set of terminals reduces to finding a minimum Steiner tree in the Hanan grid, with edge costs given by the L_1 (or Manhattan) metric, $d(u, v) = |x_u - x_v| + |y_u - y_v|$.

The IRV algorithm yields good results when applied to a graph for which the cost and structure of the minimum Steiner tree does not change much after the removal of Steiner-Steiner edges. For the RST problem, the best choice with respect to solution quality is to run IRV on the *complete* graph induced by the Hanan grid. We obtain a practical running time by applying a few simple, yet very effective, reductions to this graph.

2.3.1 Edge reductions

By a result of Robins and Salowe [66], for any set of points there exists a rectilinear MST in which each point p has at most one neighbor in each of the four diagonal quadrants, $-x \leq y < x$, $-y < x \leq y$, $x < y \leq -x$, and $y \leq x < -y$, translated at p. Hence, the optimum Steiner tree in the quasi-bipartite graph is not affected if we remove all edges incident to a Steiner vertex except those connecting it to the closest terminals from each quadrant. We can also discard all edges connecting pairs of terminals except for the $|T| - 1$ edges in MST(T)—this merely amounts to a particular choice of breaking ties between terminal-terminal edges during RV-Phase. Combined, these two edge reductions leave a quasi-bipartite graph with $O(|T| + |S|)$ edges, as opposed to $O(|T| \cdot (|T| + |S|))$ without

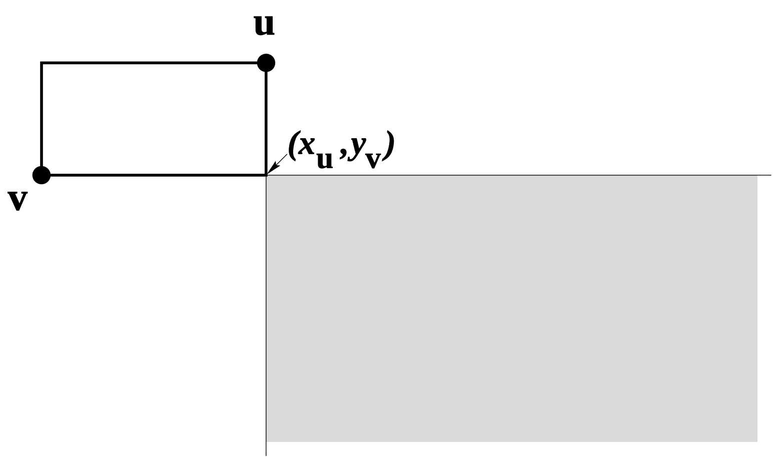

Figure 1: The empty rectangle test

edge reductions.

2.3.2 Vertex reductions

Zachariasen [76] noted that reductions based on structural properties of full Steiner components, which play a crucial role for exact algorithms such as [30] and [73], can also be used to remove from the Hanan grid a large number of Steiner vertices without affecting the optimum Steiner tree. Simpler versions of these reductions suffice in our case, since we only want to leave unaffected the optimum Steiner tree in the graph that results after the removal of Steiner-Steiner edges.

We incorporated in our code a version of the *empty rectangle* test [76], originally due to Berman and Ramaiyer [8]. Consider a grid point found, say, at the intersection of the vertical line through a terminal u and the horizontal line through a terminal v (see Figure 1). The empty rectangle test says that the point must be retained in the graph only if (1) the rectangle determined by terminals u and v is empty, i.e., contains no terminals in its interior, and (2) the shaded quadrant contains at least one terminal. We used a simple $O(|T|^2)$ implementation of this test; an $O(|T| \log |T| + k)$ implementation, where k is the number of empty rectangles, is also possible [39].

As noted in [31, 76], a stronger version of the empty rectangle test is guaranteed to remove all but a set of $O(|T|)$ Steiner points, still with no increase in the cost of the optimum RST with no Steiner-Steiner edges. By using this stronger test the overall running time of IRV as applied to RST reduces to $O(k \cdot |T|^2)$, where k is the number of crystallized Steiner vertices (usually a small fraction of $|T|$).

2.4 Experimental results

We compared our algorithm against Robins' implementation [65] of BI1S [49, 37], and against the recent release [74] of the exact GeoSteiner algorithm of Warme, Winter, and Zachariasen [73].

2.4.1 Experimental setup

All experiments were conducted on a SGI Origin 2000 with 16 195MHz MIPS R10000 processors—only one of which is actually used by the sequential implementations included in our comparison—and 4 G-Bytes of internal memory, running under IRIX 6.4 IP27. Timing was performed using low-level Unix interval timers, under similar load conditions for all experiments.

We coded our heuristic in C; Robins' BI1S implementation and GeoSteiner are coded in C as well. The three programs were compiled using the same compiler (gcc version egcs-2.90.27) and optimization options (-O4). Whenever we had a choice in the configuration of BI1S or GeoSteiner, we optimized for speed. The only exception to this rule was the choice of LP–solver in GeoSteiner: Since CPLEX was not available on our test machine, we configured GeoSteiner to use Warme's customized version of the public-domain package lp_solve. In order to assess the loss in speed induced by this choice, we ran both versions of GeoSteiner on a different machine that had a licensed copy of CPLEX 6.5.1. Although CPLEX is generally considered to be significantly faster than lp_solve, the CPLEX version of GeoSteiner was *only 30%* faster than the lp_solve version on experiments involving 1000 random instances. The lp_solve version of GeoSteiner contains some optimizations—made possible by intimate access to the internals of lp_solve—not included in the CPLEX version. However, since the optimized portions of the code were infrequently executed in our experiments, these optimizations do not fully explain the unexpectedly small speed advantage of CPLEX. As suggested by David Warme, the most plausible explanation is the expensive CPLEX preprocessing, that pays off handsomely on large and difficult instances but not as well on the LPs occurring in our computational study.

The test bed for our experiments consisted of two categories of instances:

- *Random instances:* For each instance size between 10 and 250, in increments of 10, we generated uniformly at random 1,000 instances[2] consisting of points in general position[3] drawn from a 10000×10000 grid.

- *Real VLSI instances:* To further validate our results, we ran the three competing algorithms on a set of 9 large instances extracted from two different VLSI designs.

2.4.2 Solution quality

Following the standard practice [46], we use the *percent improvement over the MST on terminals,*

$$\frac{cost(\text{MST}) - cost(\text{Algo. RST})}{cost(\text{MST})} \times 100,$$

to compare the quality of the RSTs produced by the three algorithms.

[2]Of the total of 25,000 random instances, the lp_solve based GeoSteiner exhibited numerical instability on 18. These instances could only be solved by turning on a perturbation scheme that has the effect of slowing down GeoSteiner. In the solution quality results reported below for GeoSteiner we use all 25,000 instances, since all of them could be solved to optimality in one way or another. However, in the running time results we omit these 18 instances to avoid penalizing GeoSteiner for the increased running time caused by turning on perturbations.

[3]A set of points is in general position if no two points share a common x- or y-coordinate.

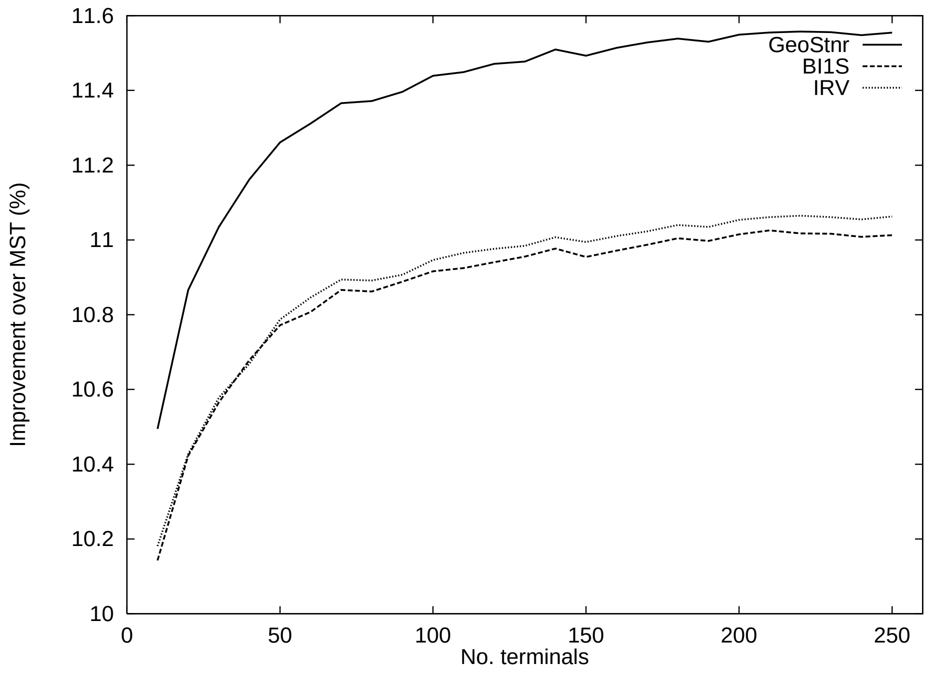

Figure 2: Average improvement over MST

Figure 2 shows the average percent improvement over MST for BI1S, IRV, and GeoSteiner on random instances. Both IRV and BI1S come on the average within 0.5% of the optimum solution found by GeoSteiner. Moreover, IRV has a very small advantage over BI1S for almost all instance sizes. Although this advantage is small—less than 0.05% on the average—it is statistically very significant, i.e., likely to be observed with high probability on any set of instances. Figure 3 shows the 95% confidence intervals for the expected difference between the percent improvement over MST of IRV and the percent improvement over MST of BI1S. For all but three instance sizes the confidence interval does not contain the origin. Wilcoxon's signed-rank sum test [42] also confirms—with a one-sided p-value lower than 0.001 for instances of size 100 or more—the small advantage that IRV has over BI1S.

For a more detailed comparison, Figure 4 gives a scatter plot of the percent improvement over MST of IRV versus that of BI1S for the 1000 random instances with 250 terminals. On

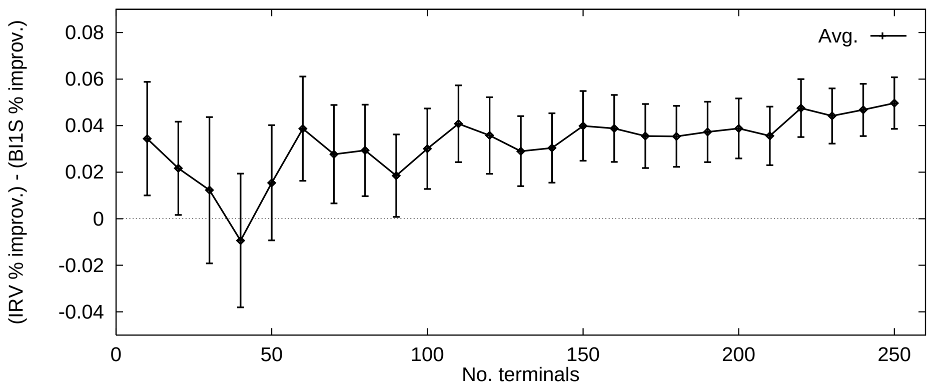

Figure 3: 95% confidence intervals for the difference in percent improvement over MST of IRV and BI1S

61% of these instances IRV finds a better solution than BI1S. The quality of the solutions produced by IRV is further illustrated by the scatter plot in Figure 5, which shows the percent improvement over MST of IRV versus the maximum possible improvement. On the same 1000 random instances, IRV is rarely more than 1% away from optimum, and on the average is less than 0.5% away.

Solution quality results on VLSI instances are presented in Table 1. These results are consistent with the findings on random instances: IRV gives solutions of the same quality as BI1S on 3 instances, of better quality on 5 instances, and of worse quality on 1 instance. Both heuristics come very close to optimum; in fact, BI1S finds an optimum solution on one instance and IRV finds optimum solutions twice.

2.4.3 Running time

We noted in Section 2.3 that our IRV implementation uses edge and vertex reductions in order to speed-up the computation. Figure 6 shows the speed-up obtained by using the empty rectangle test described in section 2.3.2; edge reductions described in Section 2.3.1 lead to an even more significant speed-up, similar to the one reported in [37] for BI1S. Despite its simplicity, the empty rectangle test reduces the number of Steiner points by a

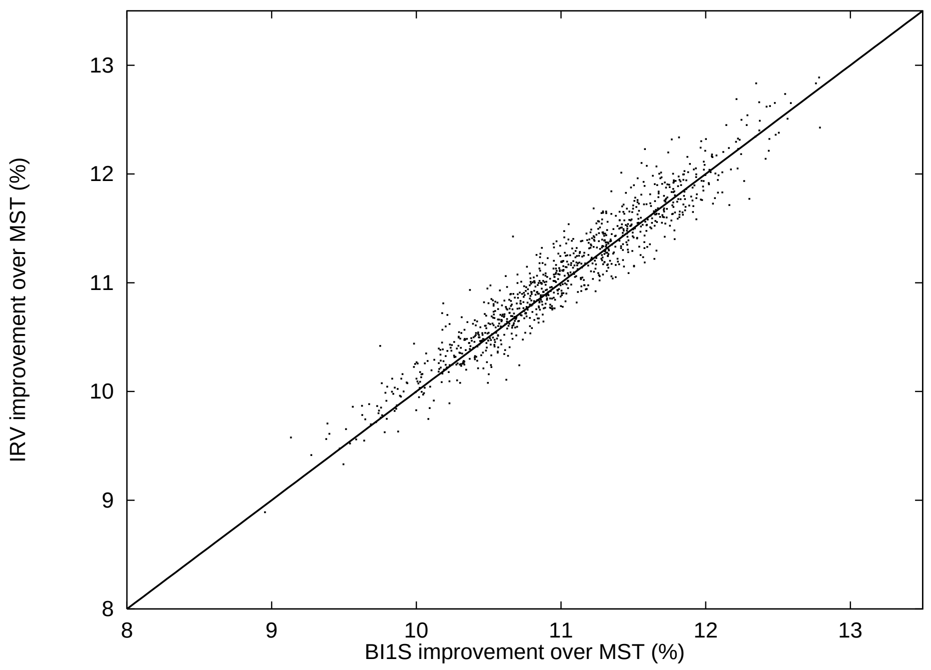

Figure 4: IRV vs. BI1S on 1000 250–terminal instances

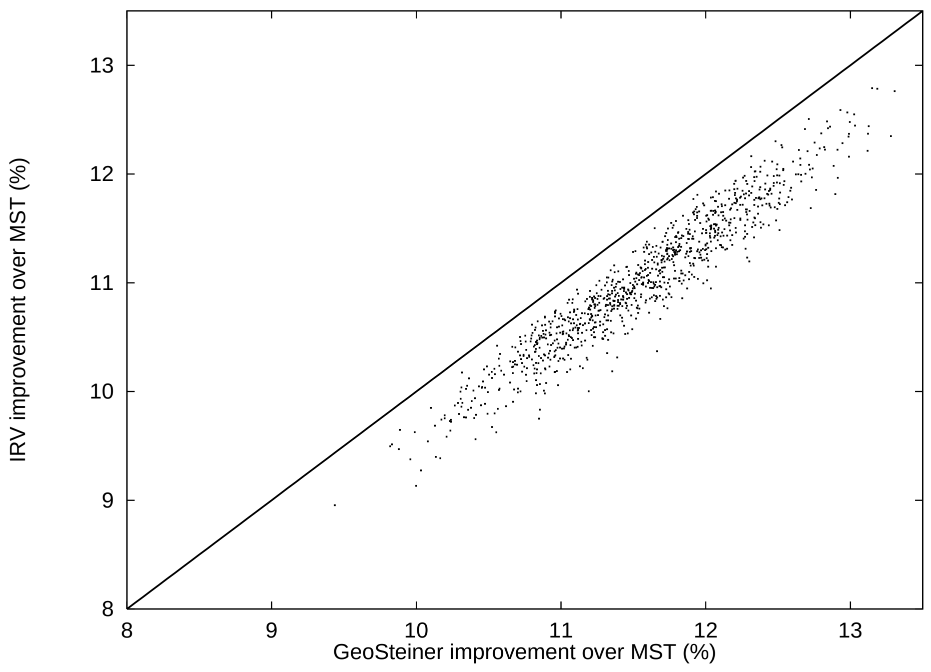

Figure 5: IRV vs. GeoSteiner on 1000 250–terminal instances

Design.Net	No. term.	No. Stnr.	BI1S	IRV	GeoStnr
16BSHREG.CLK	185	1375	1.757	1.757	1.757
16BSHREG.RESET	406	4730	3.666	3.666	3.810
16BSHREG.VDD	573	5089	8.079	8.079	8.118
16BSHREG.VSS	556	6058	7.854	8.131	8.192
MAR.BRANCH	188	2034	9.007	9.158	9.221
MAR.CLK	264	3355	7.637	7.748	7.957
MAR.GND	245	3264	6.300	6.321	6.476
MAR.RESET	109	1021	11.206	11.246	11.246
MAR.VDD	340	3681	6.038	6.003	6.181

Table 1: Average percent improvement over MST on VLSI instances

Design.Net	No. term.	No. Stnr.	BI1S	IRV	GeoStnr
16BSHREG.CLK	185	1375	1.31	0.25	2.80
16BSHREG.RESET	406	4730	10.07	1.65	4.37
16BSHREG.VDD	573	5089	30.29	2.94	1.73
16BSHREG.VSS	556	6058	36.71	3.29	7.90
MAR.BRANCH	188	2034	1.26	0.62	5.21
MAR.CLK	264	3355	2.34	1.57	13.16
MAR.GND	245	3264	1.96	1.26	1.03
MAR.RESET	109	1021	0.24	0.16	0.65
MAR.VDD	340	3681	7.69	1.59	8.19

Table 2: Running time on VLSI instances

factor of over 15 for 250 terminal instances. This speeds up the IRV algorithm by essentially the same factor. Due to the direct correspondence between the reduction in Steiner vertices and the speed-up of the algorithm, it seems worthwhile to explore further reduction ideas, in particular those suggested in [31, 76].

After noticing that BI1S can also benefit from vertex reductions, we incorporated the empty rectangle test into Robins' code. As shown in Figure 6, we obtained again a speed-up roughly equal to the decrease in the number of Steiner vertices. All running times reported below for BI1S refer to this sped-up version of Robins' code.

Figure 7 compares the average running time of BI1S, IRV, and GeoSteiner on random instances. On these instances IRV is 4–10 times faster than GeoSteiner and BI1S is 30% faster than IRV. Surprisingly, the results on VLSI instances presented in Table 2 indicate different trends than results on random instances. On these instances, both IRV and GeoSteiner run significantly faster than predicted by experiments on random instances. In particular, IRV is always faster than BI1S, sometimes by as much as a factor of 10. Although IRV is still faster than GeoSteiner, the difference in speed is not as impressive on these instances as it is on random instances.

2.4.4 Convergence rate

Figures 8 and 9 display the rate of convergence to the final solution for BI1S, IRV, and GeoSteiner, when run on 16BSHREG.RESET and on a random instance of the same size, respectively. Each point on the IRV and BI1S curves corresponds to the addition/deletion of a single Steiner point to/from the solution; the points defining the GeoSteiner curve represent moments when better feasible solutions are found during the branch-and-cut search. A logarithmic time-scale is used in both figures to better put in perspective the early rate of convergence for each algorithm.

Although Figures 8 and 9 don't give much insight on GeoSteiner, they do capture many of the fundamental features of BI1S and IRV. For example, it is immediately apparent that

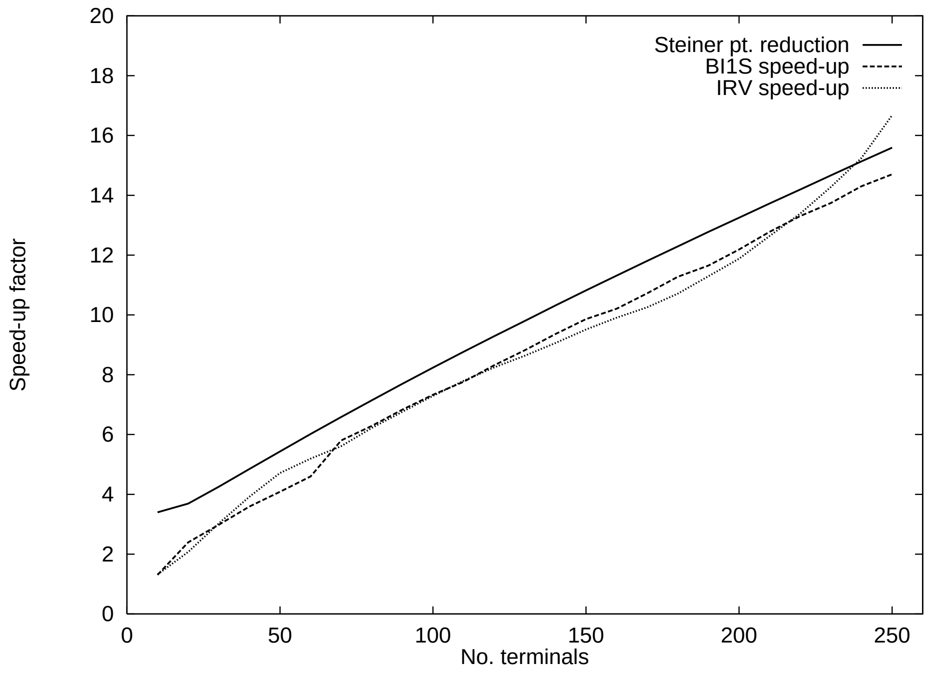

Figure 6: Speed-up achieved by using the empty rectangle test

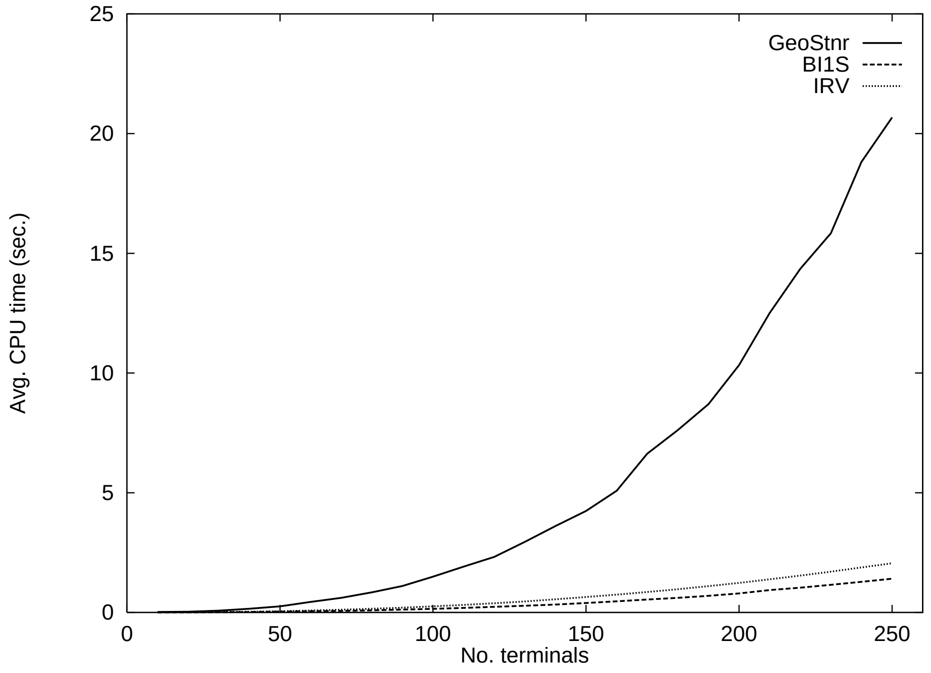

Figure 7: Average CPU time

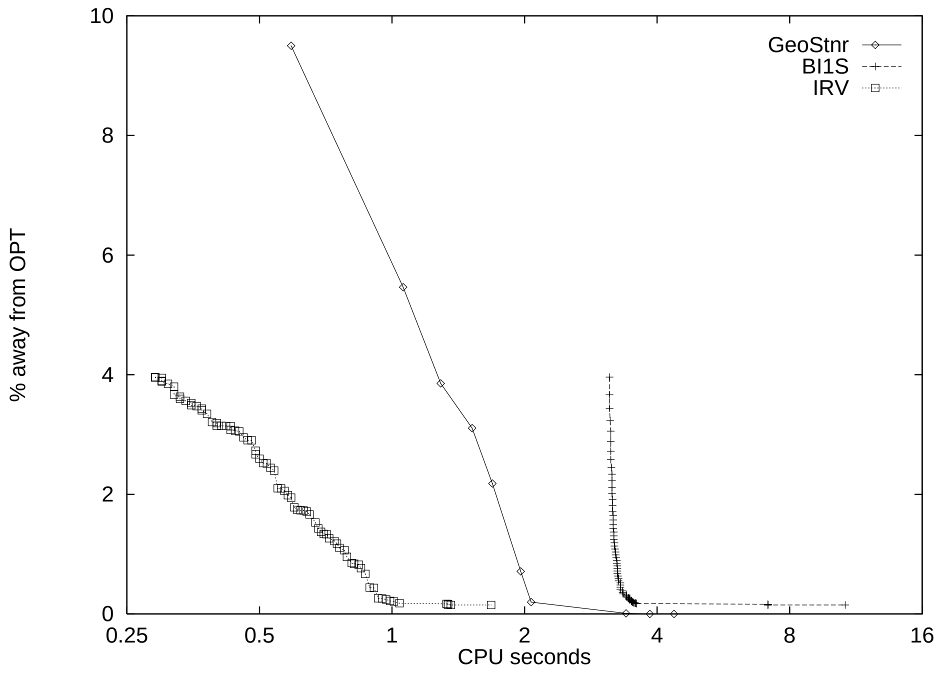

Figure 8: Convergence to the final solution on instance 16BSHREG.RESET

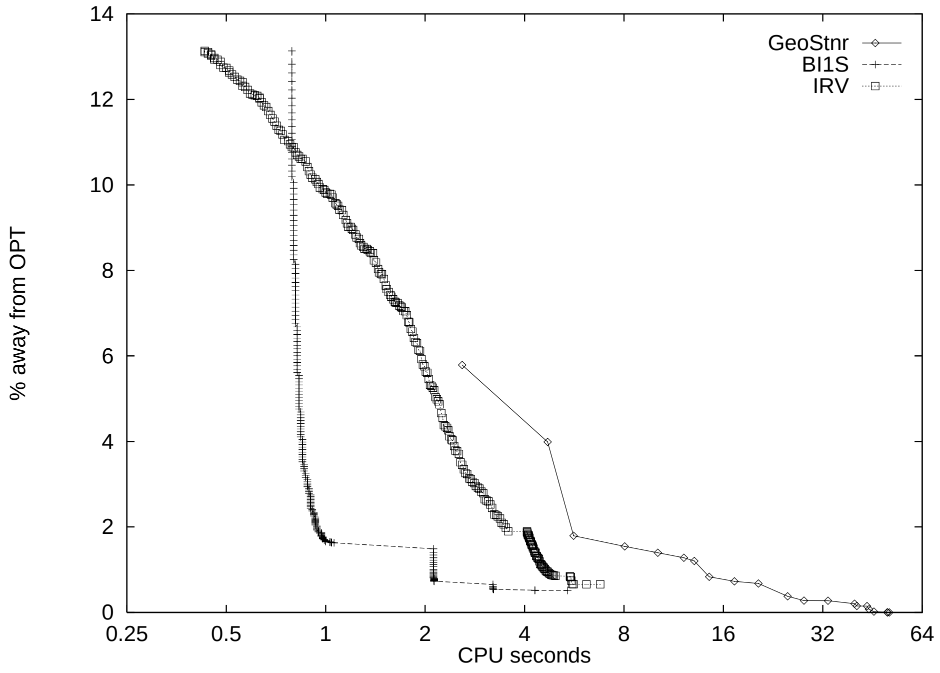

Figure 9: Convergence to the final solution on a random 406–terminal instance

BI1S uses a greedy strategy, selecting in each step the point whose addition to the solution gives the largest immediate gain. By contrast, due to the different nature of its selection rules, IRV may select some points with very large gain only late in the process. IRV differs from BI1S not only with respect to the order in which Steiner points are selected, the two algorithms end up making different selections as well. For example, although the two algorithms obtain solutions of the same cost on 16BSHREG.RESET, the two solutions are not identical. Out of 68 Steiner points selected by IRV and 67 selected by BI1S, only 51 are shared—42 of which are also selected by GeoSteiner.

The phase structure of IRV and BI1S is also clearly visible in Figures 8 and 9. Both algorithms need 3 phases on 16BSHREG.RESET, and 5 on the random instance. The first phase adds most Steiner vertices to the solution, also giving the bulk of the overall improvement in solution quality. The following phases add fewer and fewer points, with the last phase merely verifying that a local optimum has been reached.

Although Steiner point selection is slower in IRV than in BI1S (compare the slopes), IRV appears to have a smaller phase–setup cost compared to BI1S. Indeed, IRV's phase initialization consists of sorting the edges in the quasi-bipartite graph, while BI1S needs to start a phase by computing the gain corresponding to each Steiner point—this is done by an MST computation for each Steiner point. Surprisingly, Robins' MST algorithm seems to work better on random sets of points: the phase–setup time in Figure 9 is less than a third of the phase-setup time in Figure 8, despite the fact that the random instance has *more* Steiner points (7366 versus 4730 in 16BSHREG.RESET). This explains why BI1S is faster than IRV on random instances but not on VLSI instances.

2.4.5 Running time predictability

Figure 10 gives histograms for the running times of GeoSteiner, BI1S, and IRV on 1000 instances of size 250. Most striking is the heavy-tailed distribution for the running time of GeoSteiner (note the logarithmic scale along GeoSteiner's time axis). The running time of BI1S has a multi-spike distribution, determined by the number of phases—typically between 2 and 4. The running time of the IRV algorithm depends in a much smoother way on the number of phases (again between 2 and 4 most of the time) due to its reduced phase–setup cost.

2.5 Conclusions

The experimental data presented in Section 2.4 shows that IRV produces high-quality rectilinear Steiner trees, typically better than those produced by the Batched Iterated 1-Steiner heuristic. The same data shows that BI1S is significantly sped up by the addition of the empty rectangle test. With this enhancement, BI1S runs 30% faster than IRV on random instances, but not on large VLSI instances as those considered in our experiments. It should be interesting to perform extensive tests on full VLSI designs to see how the running times of the two heuristics compare when applied to a mix of both small and large nets.

Our experimental data also confirms the excellent average running time of the exact GeoSteiner algorithm of Warme, Winter, and Zachariasen [73]. When exact algorithms achieve practical running times, one is immediately prompted to ask if any interest remains in suboptimal heuristics. We think that this interest will not disappear, at least not in those RST applications where speed is more important than a small loss in solution accuracy, e.g., in wire-length estimation during placement.

Compared to GeoSteiner, heuristics such as BI1S and IRV have the advantage of a more predictable and worst-case bounded running time.[4] Moreover, BI1S and IRV hold more promise than the GeoSteiner algorithm for giving efficient solutions to objective functions other than length minimization. Since both BI1S and IRV are essentially solving the Steiner tree problem in graphs, they can be adapted without much loss in efficiency to

[4]Of course, GeoSteiner takes exponential time in the worst case. For example, Berman, Fössmeier, Karpinski, Kaufmann and Zelikovsky [7] (see also [50], pp. 39–40) give an infinite family of RST instances on which GeoSteiner provably needs exponential time.

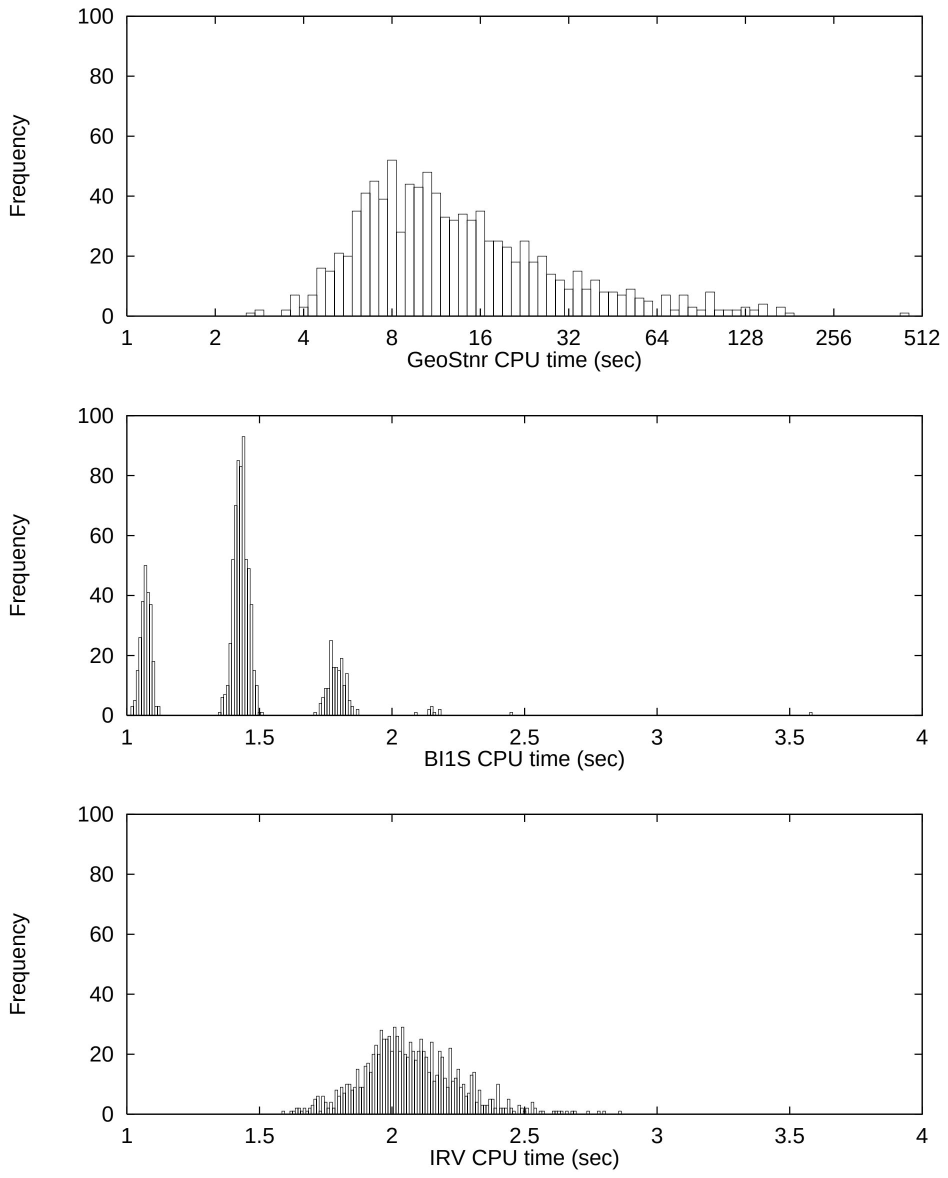

Figure 10: Histograms of running times on 1000 250–terminal instances

almost any edge-cost function—IRV does rely on costs satisfying triangle inequality. In contrast, the efficiency of a critical phase in the GeoSteiner algorithm, the Full Steiner Tree (FST) generation phase, heavily depends on structural properties specific to the underlying metric space. Even when well-understood, these structural properties may not lead to the same efficiency as in the rectilinear case. For example, FST generation is more than 100 times slower on Euclidean instances than it is on rectilinear ones, and becomes in this case the bottleneck of the whole algorithm [73].

Chapter 3

Practical approximation algorithms for zero- and bounded-skew clock trees*

3.1 Introduction

Today's high-performance VLSI circuits use almost exclusively synchronous designs. In these circuits, a clock signal, distributed by means of a tree rooted at the clock *source*, must be delivered periodically to a set of synchronizing elements, or clock *sinks*. To achieve maximum clock rate it is necessary to minimize the *clock skew*, i.e., the maximum difference in arrival times of the clock signal at synchronizing elements.

In the typical VLSI physical design cycle, clock routing is done after the placement phase has determined positions for all clock sinks. At this stage the clock skew can be controlled in a number of ways, e.g., by using wires with non-uniform width or by inserting buffers. We address the most popular approach, which is to control the *wirelength* in the clock tree. In this approach, a feasible routing for a set of sinks S is a *zero-skew tree* (ZST), i.e., a rooted Steiner tree in which all root-to-sink paths have equal length. Due to power consumption, signal integrity, and area utilization considerations, the objective is to minimize the total length of the ZST. Thus the clock tree construction problem has been formalized [5] as follows:

Rectilinear Zero-Skew Tree Problem: Given a set S of sinks in the rectilinear plane, find a zero-skew tree of minimum total length for S.

*This chapter is based on joint work with Alexander Z. Zelikovsky [78].

As noted in [50], a more realistic design requirement is captured by bounded-skew trees (BST). A rooted Steiner tree T for the set S of sinks is a *b-bounded-skew tree* if the difference in length between any two root-to-sink paths is at most b.

Rectilinear Bounded-Skew Tree Problem: Given a set S of sinks in the plane and bound $b > 0$, find a b-bounded-skew tree of minimum total length for S.

The rectilinear BST problem and the generalization of the ZST problem to arbitrary metric spaces are NP-hard [16]. The complexity of the rectilinear ZST problem is not known—for a fixed tree topology the problem can be solved in linear time by using the *Deferred-Merge Embedding* (DME) algorithm independently introduced in [10, 13, 24].

Although the rectilinear zero- and bounded-skew tree problems have received much attention in the VLSI CAD literature [6, 10, 13, 14, 19, 24, 25, 47, 48, 54] (see Chapter 4 of [50] for a detailed review), the first algorithms with constant approximation factors have been proposed only recently, by Charikar et al. [16]. Charikar et al. generalize the ZST and BST problems to arbitrary metric spaces, and, in this general setting, give algorithms with approximation factors of $2e \approx 5.44$ and 16.86, respectively. The BST algorithm in [16] relies on an approximation algorithm for the Steiner tree problem in graphs. Using the currently best Steiner tree approximation of Robins and Zelikovsky [67] and Arora's PTAS for computing rectilinear Steiner trees [3, 4], the BST bounds in [16] can be updated to 16.11 for arbitrary metric spaces, and to 12.53 for the rectilinear plane (see Table 3).

In this chapter we introduce a new approach to these problems, based on zero-skew "stretching" of spanning trees. The new approach leads to simple algorithms with improved approximation guarantees for the rectilinear ZST and BST problems, and for their extensions to arbitrary metric spaces introduced in [16]. Our contributions include:

- constructive lower bounds on the cost of the optimum ZST and BST in arbitrary metric spaces;

- improved approximation for the ZST problem in arbitrary metric spaces, based on a reduction to the *zero-skew spanning tree problem*;
- improved approximation for the ZST problem in metrically convex metric spaces,[1] based on skew elimination using Steiner points;
- improved approximation for the BST problem in arbitrary and metrically convex metric spaces, based on combining an approximate ZST with a minimum spanning tree for the sinks.

An important feature of our algorithms is their practical running time, which is asymptotically the same as the time needed for computing the minimum spanning tree. Thus, our algorithms can easily handle the clock nets with hundreds of thousands of sinks that occur in large cell-based or multi-chip module designs. For a summary of our results and a comparison to the results of Charikar et al. [16] we refer the reader to Table 3.

The rest of the chapter is organized as follows. In next section we start with a formal definition of the ZST and BST problems in general metric spaces and prove new lower bounds on the length of the optimal ZST and BST. Then, in Section 3.3, we show how to convert (or "stretch") a rooted tree T spanning the set S of sinks into a zero-skew tree for S. We show that such "stretching" increases the length by the sum of sink delays, where the *delay* in T of a sink s is the length of the path connecting s to its furthest descendant. We also show that, for metrically convex metric spaces such as the Euclidean or rectilinear planes, it is possible to reduce the "stretching" length to half the sum of delays.

In Section 3.4 we give a Kruskal-like algorithm that builds a rooted spanning tree T whose total delay does not exceed its length, and whose length is at most twice longer than that of the optimal ZST. These two facts yield an approximation factor of 4 for the ZST problem in arbitrary metric spaces and an approximation factor of 3 for metrically convex

[1]A metric space (M, d) is called *metrically convex* if, for every $u, v \in M$ and $0 \leq \lambda \leq 1$, there exists a point $w \in M$ such that $d(u, w) = \lambda d(u, v)$ and $d(w, v) = (1 - \lambda)d(u, v)$.

Problem		**Zero-Skew Tree**			**Bounded-Skew Tree**		
Metric		**General**	**Convex**	**Rectilinear**	**General**	**Convex**	**Rectilinear**
Approximation factor	[16]	$2e \approx 5.44$			16.11*		12.53*
	This thesis	4	3		14	11	9
Running time	[16]	strongly polynomial			strongly polynomial		
	This thesis	$O(n^2)$		$O(n \log n)$	$O(n^2)$		$O(n \log n)$

Table 3: Summary of results on the zero- and bounded-skew tree problems and comparison to the results of Charikar et al.

metric spaces. In Section 3.5 we discuss the implications of combining our ZST heuristics with the DME algorithm.

Finally, in Section 3.6, we describe a construction of approximate bounded-skew trees based on combining an approximate zero-skew tree for a subset of the sinks with subtrees of a minimum spanning tree (MST) or approximate minimum Steiner tree for the sinks. Combination with the MST gives a 14-approximation algorithm for the bounded-skew tree problem in arbitrary metric spaces; the factor is reduced to 11 for arbitrary metrically convex metric spaces, and to 9 for the rectilinear plane.

3.2 Constructive lower bounds

In this section, we establish new lower bounds for the ZST and BST problems in an arbitrary metric space; in contrast to the lower bounds of Charikar et al. [16] these bounds are constructive. We start by formalizing the "stretching" alluded to in the introduction and defining the ZST and BST problems in an arbitrary metric space.

Let (M, d) be an arbitrary metric space. A *stretched tree* $T = (V, E, \pi, cost)$ for a set of sinks $S \subseteq M$ is a rooted tree with node set V and edge set E, together with a pair of

*Values updated with respect to [16] by taking into account the currently best Steiner tree approximation of Robins and Zelikovsky [67], respectively Arora's PTAS for computing rectilinear Steiner trees [3, 4].

mappings, $\pi : V \to M$ and $cost : E \to I\!R_+$, such that

(1) π is a 1–1 mapping between the leaves of T and S, and

(2) for every edge $(u, v) \in E$, $cost(u, v) \geq d(\pi(u), \pi(v))$.

We refer to $\pi(u)$ as the *embedding* of u. A stretched tree T is a *zero-skew tree* if all root-to-leaf paths in T have equal cost; T is a *b-bounded-skew tree* if the difference between the cost of any two root-to-leaf paths is at most b.

Zero-Skew Tree Problem: Given a set of sinks S in metric space (M, d), find a minimum cost zero-skew tree for S.

Bounded-Skew Tree Problem: Given a set of sinks S in metric space (M, d) and bound $b > 0$, find a minimum cost b-bounded-skew tree for S.

The minimum cost of a ZST (BST) for S will be denoted by $ZST^*(S)$, respectively $BST^*(S)$. In our analysis we will use the following constructive lower bound on $ZST^*(S)$:

Lemma 5 *Let S be a set of n sinks. Then, for any enumeration $s_1, s_2, \ldots, s_n$ of the sinks in S,*

$$ZST^*(S) \geq MinDist\{s_1, s_2\} + \frac{1}{2}\sum_{i=2}^{n-1} MinDist\{s_1, \ldots, s_{i+1}\}$$

where $MinDist\{A\} = \min_{u,v \in A,\, u \neq v} d(u, v)$.

Proof : For any $r \geq 0$, let $N(r)$ denote the minimum number of closed balls of (M, d) needed to cover all sinks in S. The radius R of S is the smallest r for which $N(r) = 1$. Charikar et al. [16] established that

$$ZST^*(S) \geq \int_0^R N(r)\,\mathrm{d}r$$

Let $r_i = MinDist\{s_1, \ldots, s_{i+1}\}/2$ for every $i = 1, \ldots, n-1$, and $r_n = 0$. Clearly, $R \geq r_1 \geq r_2 \geq \cdots \geq r_{n-1} \geq r_n$. Note that $N(r) \geq i+1$ for $r < r_i$, since no two points in the set $\{s_1, \ldots, s_{i+1}\}$ can be covered by the same ball of radius r. Hence,

$$\int_0^R N(r)\,\mathrm{d}r \geq \sum_{i=1}^{n-1} \int_{r_{i+1}}^{r_i} (i+1)\,\mathrm{d}r = \sum_{i=1}^{n-1} (i+1)(r_i - r_{i+1}) = 2r_1 + \sum_{i=2}^{n-1} r_i$$

and the lemma follows. $\square$

It can be shown that the greedy enumeration (e.g. start from a diametrical pair of points and add each time the point maximizing minimum distance to previously enumerated points) may not deliver the maximum to the lower bound established in Lemma 5. The complexity of finding the best enumeration is an open question.

Below we bound the cost of the optimum BST by comparing it with the optimum ZST.

Lemma 6 *Let S be a set of sinks. Then, for any $W \subseteq S$ and skew bound $b > 0$,*

$$BST^*(S) \geq ZST^*(W) - b \cdot (|W| - 1)$$

Proof : Let T be a b-bounded-skew tree for S. We use T to construct a ZST for W of cost no larger than $cost(T) + b \cdot (|W| - 1)$ as follows. First, notice that T contains a b-bounded-skew tree for W, say T', as subtree. Let P_u denote the unique path in T' connecting u to the root, and let u_0 be a leaf of T' for which $cost(P_{u_0})$ is maximum. We get a zero-skew tree for W by adding to T' a *loop*, i.e., an edge whose ends are embedded at the same point, of cost $cost(P_{u_0}) - cost(P_u)$ for each leaf $u \neq u_0$. Since T' has skew at most b, each of the $|W| - 1$ added loops has cost at most b. Thus, the resulting ZST has cost at most $cost(T') + b \cdot (|W| - 1) \leq BST^*(S) + b \cdot (|W| - 1)$. $\square$

3.3 Zero-skew stretching of spanning trees

Let $T = (S, E)$ be a rooted tree spanning the sinks in $S \subseteq M$. For any sink u, let T_u denote the subtree of T rooted at u. The *delay* in T of u is defined by

$$delay_T(u) = \text{max}\{length(P_{uv}) \mid v \text{ leaf in } T_u\}$$

where P_{uv} denotes the unique path in T connecting u and v, and $length(P_{uv}) = \sum_{e \in P_{u,v}} d(e)$.

Let $length(T) = \sum_{e \in E} d(e)$ and $delay(T) = \sum_{u \in S} delay_T(u)$. In this section we show that, for an arbitrary metric space (M, d), T can be stretched to a zero-skew tree of cost $length(T) + delay(T)$. The stretched zero-skew tree uses no Steiner points, i.e., has all nodes embedded at the sinks. We also show that, by using Steiner points, the amount of stretching can be reduced to half the delay of T in case the underlying space is metrically convex.

3.3.1 Zero-skew stretching in arbitrary metric spaces

The stretching algorithm for arbitrary metric spaces (Algorithm 4) replaces each node u of T by by $k + 1$ nodes, $u_0, u_1, \ldots, u_k$, all embedded at u, where k is the degree of u in T. Each child v of u in T is attached to a distinct copy of u (see Figure 11) by an edge of cost equal to $d(u, v)$, which is the length of the edge (u, v) in T. The $k + 1$ copies of u are connected by loops in a path of total cost $delay_T(u)$, and the cost of each loop is set so that all paths connecting u_k to leaves of the stretched subtree have the same cost. For clarity, in Algorithm 4 we omit curly braces for single element sets and use "$-$" and "+" instead of "$\setminus$" and "$\cup$", respectively.

Lemma 7 *The stretched tree produced by Algorithm 4 has zero-skew and total cost equal to* $length(T) + delay(T)$.

Proof : It is easy to see that the algorithm produces a zero-skew tree. Since the algorithm assigns a cost of $d(u, v_k) + delay_T(v_k) = delay_T(u)$ to the path $(u_k, u_{k-1}, \ldots, u_1, u_0)$, the

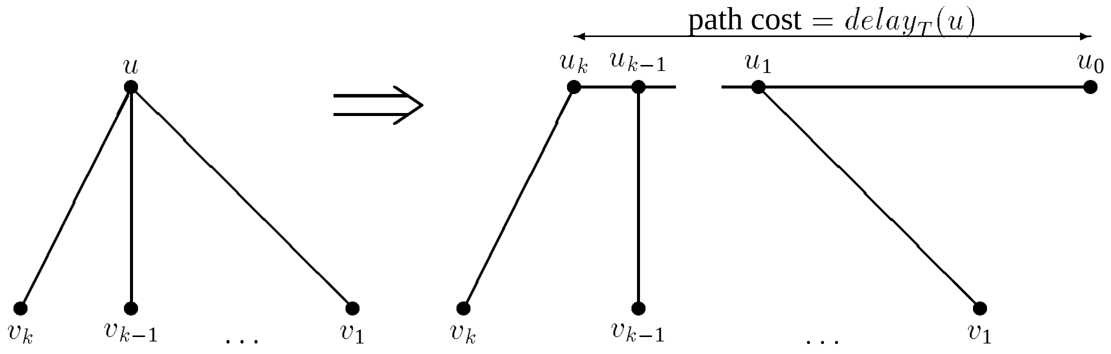

Figure 11: The basic step of the stretching algorithm for arbitrary metric spaces

```
Input: Rooted spanning tree $T = (S, E)$ in a metric space $(M, d)$
Output: Zero-skew tree $T_1 = (V_1, E_1, \pi, cost)$ for $S$

1. $V_1 \leftarrow S$, $\pi(v) \leftarrow v$ for any $v \in V_1$
2. $E_1 \leftarrow E$, $cost(u, v) \leftarrow d(u, v)$ for any $(u, v) \in E_1$
3. For each sink $u \in S$, do:

    Sort $u$'s children, say $v_1, v_2, \dots, v_k$, such that
        $d(u, v_1) + delay_T(v_1) \le d(u, v_2) + delay_T(v_2) \le \cdots \le d(u, v_k) + delay_T(v_k)$
    // Replace each node $u$ with $k + 1$ nodes embedded at $u$
    $V_1 \leftarrow V_1 - u + \{u_0, \dots, u_k\}$,
    $\pi(u_0) \leftarrow \pi(u_1) \leftarrow \cdots \leftarrow \pi(u_k) \leftarrow u$
    // Replace edges $(u, v_i)$ with edges through copies of $u$
    $E_1 \leftarrow E_1 - (u, v_k) + (u_k, v_k)$;   $cost(u_k, v_k) \leftarrow d(u, v_k)$
    For $i = 1, \dots, k - 1$ do
        $E_1 \leftarrow E_1 - (u, v_i) + (u_{i+1}, u_i) + (u_i, v_i)$;
        $cost(u_i, v_i) \leftarrow d(u, v_i)$;
        $cost(u_{i+1}, u_i) \leftarrow [\ d(u, v_{i+1}) + delay_T(v_{i+1})\ ] - [\ d(u, v_i) + delay_T(v_i)\ ]$
    $E_1 \leftarrow E_1 + (u_0, u_1)$;   $cost(u_0, u_1) \leftarrow d(u, v_1) + delay_T(v_1)$
4. Output $T_1 = (V_1, E_1, \pi, cost)$
```

Algorithm 4: The zero-skew stretching algorithm for arbitrary metric spaces

cost of T_1 increases by

$$\left(\sum_{i=1}^{k} d(u, v_i)\right) + delay_T(u)$$

when processing node u of T in Step 3 of the algorithm. Hence, the total cost of T_1 is $length(T) + delay(T)$. □

3.3.2 Zero-skew stretching in metrically convex metric spaces

Before stating the algorithm, we need to introduce some more notations. A path $P = (p_1, p_2, \ldots, p_k)$ in T_1 is called *critical* if p_k is a leaf of T_1 and $cost_{T_1}(P) = length(P)$. By construction, it follows that the tree T_1 produced by Algorithm 4 has at least one critical path starting from each node. Let $P = (p_1, p_2, \ldots, p_k)$ be a critical path in T_1. For every $0 \leq \delta \leq length(P)$, there exist i such that $length(p_1, p_2, \ldots, p_i) \leq \delta < length(p_1, p_2, \ldots, p_{i+1})$. We denote the edge (p_i, p_{i+1}) by $e(P, \delta)$. Since (M, d) is metrically convex, there is a point $v(P, \delta) \in M$ such that such that the $length(p_1, \ldots, p_i, v(P, \delta)) = \delta$ and $length(v(P, \delta), p_{i+1}, \ldots, p_k) = length(P) - \delta$.

The improved stretching in metrically convex metric spaces (Algorithm 5) is based on the following observation: any loop in the stretched tree T_1 produced by Algorithm 4 can be "folded" along a critical path of T_1, thus saving half of the cost of the loop (see Figure 12).

Lemma 8 *The stretched tree T_2 produced by Algorithm 5 has zero-skew and total cost equal to $length(T) + delay(T)/2$.*

Proof : The total cost of the loops in the stretched tree T_1 is equal to $delay(T)$. Step 3 of the algorithm folds all these loops, saving half from the cost of each (see Figure 12). Therefore, $cost(T_2) = length(T) + delay(T)/2$. The tree T_2 has zero-skew since T_1 has zero-skew and loop folding preserves all root-to-leaf path costs. □

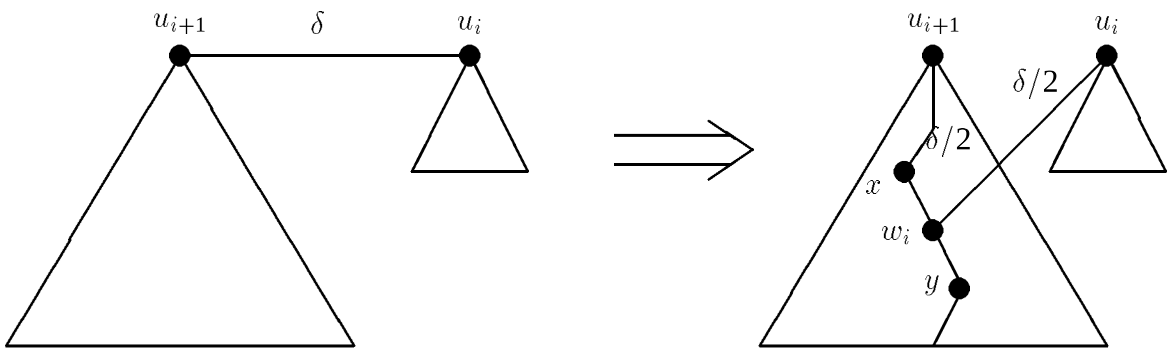

Figure 12: Loop folding in metrically convex metric spaces

```
Input: Rooted spanning tree $T = (S, E)$ in a metric space $(M, d)$
Output: Zero-skew tree $T_2 = (V_2, E_2, \pi, cost)$ for $S$
--------------------------------------------------------------------
1. Find $T_1 = (V_1, E_1, \pi, cost)$ using Algorithm 4
2. $(V_2, E_2, \pi, cost) \leftarrow (V_1, E_1, \pi, cost)$
3. For each loop $(u_{i+1}, u_i) \in E_2$ do
       // Add the attachment point to the critical path from $u_{i+1}$
       Find edge $(x, y) = e(P, \delta/2)$ on the critical path $P$ from $u_{i+1}$, where
           $\delta = cost(u_{i+1}, u_i)$
       $V_2 \leftarrow V_2 + w_i$, where $w_i = v(P, \delta/2)$
       $E_2 \leftarrow E_2 - (x, y) + (x, w_i) + (w_i, y)$;  $cost(x, w_i) \leftarrow d(x, w_i)$;  $cost(w_i, y) \leftarrow d(w_i, y)$
       // Replace the loop $(u_{i+1}, u_i)$ with the edge $(w_i, u_i)$
       $E_2 \leftarrow E_2 - (u_{i+1}, u_i) + (w_i, u_i)$;  $cost(w_i, u_i) \leftarrow \delta/2$
4. Output $T_2 = (V_2, E_2, \pi, cost)$
```

Algorithm 5: The zero-skew stretching algorithm for metrically convex metric spaces

3.4 ZST approximation via spanning trees

In the previous section we have shown that one can stretch any rooted spanning tree into a zero-skew tree whose cost is equal to the length of the spanning tree plus its delay (half the delay, for metrically convex metric spaces). This motivates the following:

Zero-Skew Spanning Tree Problem: Given a set of points S in a (metrically convex) metric space (M, d), find a rooted spanning tree T on S such that $cost(T) = length(T) + delay(T)$ (respectively, $length(T) + delay(T)/2$) is minimized.

Note that the minimum spanning tree (MST) on S has the shortest possible length but may have very large delay—if the MST is a simple path, then its delay may be as much as $O(n)$ times larger than its length. On the other hand, a star having the least delay may be $O(n)$ times longer than the MST.

In this section we give an algorithm for finding a rooted spanning tree which has both delay and length at most two times the minimum ZST cost. Therefore, our algorithm gives factor 4 and 3 approximations for the ZST problem in general and metrically convex metric spaces, respectively. Simultaneously, our algorithm gives factor 4 and 3 approximations for the zero-skew spanning tree problem in the respective metric spaces, since $cost(T)$ cannot be smaller than the cost of the minimum ZST.

The algorithm (Algorithm 6) can be thought of as a rooted version of the well-known Kruskal MST algorithm. At all times, the algorithm maintains a collection of rooted trees spanning the sinks, initially each sink is a tree by itself. In each step, the algorithm chooses two trees that have the smallest distance between their roots and merges them by linking the root of one tree as child of the other. In order to keep the delay of the resulting tree small, the child root is always chosen to be the root with smaller delay.

Lemma 9 $delay(T) \leq length(T)$

```
Input: Finite set $S \subseteq M$
Output: Rooted spanning tree $T$ on $S$
--------------------------------------------------------------
1. Initialization:
    $ROOTS \leftarrow S$; $E \leftarrow \emptyset$
    For each $v \in S$, $h(v) \leftarrow 0$
2. While $|ROOTS| > 1$ do:
    Find the closest two sinks $r, r' \in ROOTS$ with respect to metric $d$
    If $h(r) < h(r')$ then swap $r$ and $r'$
    $E \leftarrow E + (r, r')$
    $h(r) \leftarrow \max\{h(r),\ d(r, r') + h(r')\}$
    $ROOTS \leftarrow ROOTS - r'$
3. Output the tree $T = (S, E)$, rooted at the only remaining sink in $ROOTS$
```

Algorithm 6: The Rooted-Kruskal algorithm

Proof : Note that, at the end of the Rooted-Kruskal algorithm, $h(u)$ represents exactly the delay of node u in T. In any iteration the algorithm adds edge (r, r') to $E(T)$, thus increasing $length(T)$ by $d(r, r')$. On the other hand, since $h(r) \geq h(r')$ when $h(r)$ is updated, the iteration contributes at most $d(r, r') + h(r') - h(r) \leq d(r, r')$ to $\sum_{u \in S} h(u)$, hence, to the total delay of T. □

Let n be the number of sinks in S.

Lemma 10 $length(T) \leq 2(1 - 1/n)ZST^*(S)$

Proof : Let s_1 be the root of T, and let $s_2, \ldots, s_n$ be the remaining $n - 1$ nodes of T, indexed in reverse order of their deletion from $ROOTS$. Since in each iteration the algorithm adds to T the edge joining a closest pair of points in $ROOTS$,

$$length(T) = \sum_{i=1}^{n-1} MinDist\{s_1, \ldots, s_{i+1}\}$$

Thus, by Lemma 5,

$$length(T) \leq 2\ ZST^*(S) - MinDist\{s_1, s_2\} = 2\ ZST^*(S) - d(s_1, s_2)$$

Since (s_1, s_2) is the longest edge in T, $d(s_1, s_2) \geq length(T)/(n-1)$, and the lemma follows. □

Lemmas 7, 9, and 10 give:

Theorem 11 *For any metric space and any set of n sinks, running Algorithm 4 on the tree T produced by the Rooted-Kruskal algorithm gives a zero-skew tree whose cost is at most $4(1-1/n)$ times larger than $ZST^*(S)$.*

Proof : By Lemma 7, the cost of the embedding is equal to $length(T) + delay(T)$. But $delay(T) \leq length(T)$ by Lemma 9, and the approximation factor follows from Lemma 10. □

Similarly, Lemmas 8, 9, and 10 give:

Theorem 12 *For any metrically convex metric space and any set of n sinks, running Algorithm 5 on the tree T produced by the Rooted-Kruskal algorithm gives a zero-skew tree whose cost is at most $3(1-1/n)$ times larger than $ZST^*(S)$.*

Proof : By Lemma 8, the cost of the embedding is now equal to $length(T) + (1/2) \cdot delay(T)$, and the theorem follows again from Lemmas 9 and 10. □

The following example shows that the algorithm in Theorem 11 can produce zero-skew trees which are $4(1-1/n)$ times larger than optimal. A similar example shows that the algorithm in Theorem 12 has a tight approximation factor of $3(1-1/n)$.

Example 13 Consider a discrete metric space on $2^k + 1$ points, $n = 2^k$ of which are sinks. We label the sinks with 0-1 sequences of length k, i.e., $S = \{\alpha = b_{k-1}b_{k-2}\dots b_0 \mid b_i \in \{0,1\}\}$. All sink-to-sink distances are equal to 1 and the distance from the single Steiner point to each of the sinks is $1/2$. In this space, the optimal ZST is a star rooted at the Steiner point, and has cost equal to $n/2$. The Rooted-Kruskal algorithm may construct the spanning tree T with root $(11\dots 1)$ and edges (α, α'), such that α' is identical to α except that the rightmost 0 in α' is replaced with 1 in α. Indeed, at each iteration of Step 2, the algorithm may choose to merge trees rooted at α and α' as above. It may choose α to be the root of the merged tree since $h(\alpha) = h(\alpha')$.

Clearly, $length(T) = n - 1$. On the other hand, since we always merge two roots with the same h-value, each merge contributes exactly 1 to the total delay of T. Thus, $delay(T) = n - 1$. By Lemma 7, the cost of the ZST produced by the algorithm is

$$length(T) + delay(T) = 2(n-1) = 4\,(1 - 1/n) \cdot \frac{n}{2}$$

□

Running time. The running time of the stretching algorithms given in Section 3.3 is dominated by the time needed to sort the children of each node; this can be done in $O(n \log n)$ overall. For arbitrary metrics the Rooted-Kruskal algorithm can be implemented in $O(n^2)$ time using Eppstein's dynamic closest-pair data structure [26]. In the rectilinear plane (in fact, in any fixed dimensional L_p space), the running time can be reduced to $O(n \log n)$ time by using the dynamic closest-pair data structure of Bespamyatnikh [9]. These implementations of the Rooted-Kruskal algorithm are asymptotically optimal, since the running times match known lower bounds for computing the first closest pair.

Finally, the total time for running the Rooted-Kruskal algorithm followed by one of the stretching algorithms given in Section 3.3 is $O(n^2)$ in arbitrary metric spaces, respectively $O(n \log n)$ in the rectilinear plane. Notice that this matches asymptotically the time needed for computing a minimum spanning tree.

3.5 Practical considerations for approximating the rectilinear ZST

In the previous two sections it has been shown how to approximate ZST in metrically convex metric spaces within a factor of 3. In order to obtain shorter ZSTs in the rectilinear plane, we may combine the stretched spanning tree with the DME algorithm [10, 13, 24]. The DME algorithm gives the optimal rectilinear ZST for any given *topology*, which is an unweighted binary tree with the leaves labeled by the sinks. Therefore, we may only shorten the rectilinear ZST if we feed the topology of the stretched spanning tree into the DME algorithm.

In Section 3.3 we suggested two different ways of stretching a spanning tree. One may expect that the topology produced by Algorithm 5 (the loop folding algorithm) is superior to the topology produced by Algorithm 4. Surprisingly, when stretching the spanning tree produced by the Rooted-Kruskal algorithm, both algorithms lead to the same topology. As proven below, in every loop folding step the attachment point for vertex u_i, $w_i = v(P, cost(u_{i+1}, u_i)/2)$, belongs to the edge (u_{i+1}, v_{i+1}) (see Figures 11 and 12), i.e., folding loops does not change the topology of the stretched tree produced by Algorithm 4.

Theorem 14 *Let T be a rooted spanning tree constructed by the Rooted-Kruskal algorithm. In any metrically convex metric space, the topologies produced by running Algorithms 4 and 5 on T are identical.*

Proof : Let the children $\{v_1, \ldots, v_k\}$ of a node u be sorted as in Algorithm 4, i.e., in non-decreasing order of $d(u, v_i) + delay_T(v_i)$. For brevity, denote $d_i = d(u, v_i)$ and $D_i = delay_T(v_i)$. We will show that $\delta = cost(u_{i+1}, u_i)$ is no greater than d_{i+1}. This will ensure that the point $v(P, \delta/2)$ lies on the edge (u_{i+1}, v_{i+1}) and, therefore, the tree topologies produced by the two stretching algorithms are the same. Since $\delta = (d_{i+1} + D_{i+1}) - (d_i + D_i)$, it suffices to prove that

$$D_{i+1} \leq d_i + D_i \tag{4}$$

We say that index k ***precedes*** index l if the node v_k has been attached to u before v_l in the Rooted-Kruskal algorithm. Let p_1 be the maximum index preceding $i+1$, p_2 be the maximum index preceding p_1, and so on, until we arrive at an index p_m with $D_{p_m} = 0$. Then $d_1 + D_1$ represents the length of the critical path from u at the time when v_{i+1} is linked to u by the Rooted-Kruskal algorithm, and $d_{p_{i+1}} + D_{p_{i+1}}$ is the length of the critical path from u at the time when v_{p_i} is linked to u.

Notice that, since $d_l \geq d_k$ if k precedes l,

$$d_{i+1} \geq d_{p_1} \geq \cdots \geq d_{p_m} \tag{5}$$

Moreover,

$$D_{i+1} \leq d_{p_1} + D_{p_1} \tag{6}$$

and

$$D_{p_{j-1}} \leq d_{p_j} + D_{p_j} \tag{7}$$

for every $j = 2, \ldots, m-1$, since through all attachments node u remains the root.

Assume, for a contradiction, that (4) does not hold. We will show by induction on j that $p_j > i+1$ and $D_{i+1} \leq D_{p_j}$ for every $j = 1, \ldots, m$. Since $D_{p_m} = 0$, the above claim implies that $D_{i+1} = 0$, making (4) trivially true.

To prove the claim, consider first $j = 1$. If $p_1 \leq i$, then $d_{p_1} + D_{p_1} \leq d_i + D_i$, and (6) implies (4). So, it must be the case that $i + 1 < p_1$. Then $d_{i+1} + D_{i+1} \leq d_{p_1} + D_{p_1}$, and (5) implies that $D_{i+1} \leq D_{p_1}$.

Assume now that $D_{i+1} \leq D_{p_{j-1}}$ for some $j \geq 2$. If $p_j \leq i$, using (7) we get

$$D_{i+1} \leq D_{p_{j-1}} \leq d_{p_j} + D_{p_j} \leq d_i + D_i$$

So, it must be the case that $i + 1 < p_j$. Then $d_{i+1} + D_{i+1} \leq d_{p_j} + D_{p_j}$ and, since $d_{i+1} \geq d_{p_j}$ by (5), this implies that $D_{i+1} \leq D_{p_j}$. □

Corollary 15 *Combination of the Rooted-Kruskal algorithm with the stretching algorithm for arbitrary metric spaces (Algorithm 4) and with the DME algorithm gives a 3-approximation for the rectilinear ZST problem.*

3.6 Approximate bounded-skew trees

In this section we give two approximation algorithms for the BST problem, both built around a black-box ZST approximation algorithm. In both cases we construct a ZST for an appropriately chosen subset of the sinks, then extend this ZST to a b-bounded-skew tree for all sinks. In first algorithm (Algorithm 7) the extension is done by adding subtrees of an MST on the sinks; in second (Algorithm 8) subtrees are extracted from an approximate Steiner tree.

3.6.1 The MST based algorithm

The first algorithm (Algorithm 7) uses a simple iterative construction to cover the sinks by disjoint b-skew subtrees of an MST T_0 of S. The algorithm then outputs the union of these subtrees with a ZST T_1 on their roots. Clearly the resulting tree T' is a b-bounded-skew tree for S. Moreover, $cost(T') \leq cost(T_1) + length(T_0)$, since the subtrees are disjoint pieces of T_0. Hence, if the ZST algorithm used in Step 3 has an approximation factor of r_{ZST}, by Lemma 6 we get that

$$\begin{aligned} cost(T') &\leq r_{ZST} ZST^*(W) + length(T_0) \\ &\leq r_{ZST}(BST^*(S) + b \cdot (|W| - 1)) + length(T_0) \end{aligned}$$

Notice that a path of length b or more is deleted from T for each node—except, possibly, the last one—added to W in Step 2 of Algorithm 7. Hence, $b \cdot (|W| - 1) \leq length(T_0)$, and so

$$cost(T') \leq r_{ZST} BST^*(S) + (r_{ZST} + 1) length(T_0)$$

```
Input: Finite set $S \subseteq M$, bound $b > 0$
Output: $b$-bounded-skew tree for $S$
--------------------------------------------------------------
1. Find an MST $T_0$ on $S$, with respect to the metric $d$, and root it at an arbitrary
   node
2. Find a set $W$ of sinks and a collection of subtrees of $T_0$, $(B_u)_{u \in W}$, as follows:
     $W \leftarrow \emptyset$; $T \leftarrow T_0$
     While $T \neq \emptyset$ do:
         Find a leaf $v$ of $T$ which is furthest from the root
         Find the highest ancestor, say $u$, of $v$ that still has $delay_T(u) \leq b$
         $W \leftarrow W \cup \{u\}$;  $B_u \leftarrow T_u$;  $T \leftarrow T \setminus B_u$
3. Find an approximate zero-skew tree, $T_1$, for $W$
4. Output the tree $T' = T_1 \cup (\bigcup_{u \in W} B_u)$
```

Algorithm 7: The MST based bounded-skew tree algorithm

Let r_{MST} be the *Steiner ratio* for the metric space (M, d), i.e., the supremum, over all sets of points S in (M, d), of the ratio between the length of an MST and the length of a minimum Steiner tree for S. Since the length of the minimum Steiner tree for S is a lower bound on $BST^*(S)$, we get that $length(T_0) \leq r_{MST} BST^*(S)$. Hence, we have the following:

Theorem 16 *Algorithm 7 has an approximation factor of* $r_{ZST} + r_{MST} + r_{ZST} r_{MST}$.

Since the Steiner ratio is at most 2 for any metric space [52], and 3/2 for the rectilinear plane [44], by using the results in Theorems 11 and 12 we get:

Corollary 17 *The approximation factor of Algorithm 7 is 14 in arbitrary metric spaces, 11 in arbitrary metrically convex metric spaces, and 9 in the rectilinear plane.*

Notice that the running time of Algorithm 7 is still $O(n \log n)$ for the rectilinear plane and $O(n^2)$ for arbitrary metric spaces: The MST in Step 1 can be computed whithin

```
Input: Finite set $S \subseteq M$, bound $b > 0$
Output: $b$-bounded-skew tree for $S$

1. Find an approximate Steiner tree $T_0$ on $S$, with respect to the metric $d$
2. Find a set $W$ of sinks and a collection of subtrees of $T_0$, $(B_u)_{u \in W}$, as follows:
      $W \leftarrow \emptyset$; $T \leftarrow T_0$
      While $T \neq \emptyset$ do:
          Pick an arbitrary sink $u$ in $T$, and let $B_u$ be the subtree of $T$ induced by
            vertices within tree distance of at most $b$ from $u$
          $W \leftarrow W \cup \{u\}$;  $T \leftarrow T \setminus B_u$
3. Find an approximate zero-skew tree, $T_1$, for $W$
4. Output the tree $T' = T_1 \cup (\bigcup_{u \in W} B_u)$
```

Algorithm 8: The approximate Steiner tree based bounded-skew tree algorithm

these time bounds using Hwang's [45] rectilinear MST algorithm and Kruskal's algorithm respectively, while Step 2 can be implemented in linear time.

3.6.2 The approximate Steiner tree based algorithm

The second BST algorithm combines a ZST for a subset W of the sinks with b-skew subtrees of an approximate Steiner tree T_0 (Algorithm 8).

Theorem 18 *The BST problem can be approximated within a factor of $r_{ZST} + r_{SMT} + 2\ r_{ZST} r_{SMT}$, given r_{ZST}, respectively r_{SMT}, approximation algorithms for the ZST and minimum Steiner tree problems.*

Proof : By construction, the distance in T_0 between any two sinks in W is at least b. Consider the set of open balls of radius $b/2$ centered at the sinks in W, with the balls considered in the metric space induced by T_0. Since any two such balls are disjoint, and each of them must cover at least $b/2$ worth of edges of T_0, we get that

$$b|W| \leq 2\ length(T_0) \tag{8}$$

To estimate the cost of the BST produced by the algorithm, notice that $\bigcup_{u \in W} B_u$ has total cost of at most $length(T_0)$. By Lemma 6 and (8), we get:

$$\begin{aligned} cost(T') &\leq r_{ZST} ZST^*(W) + length(T_0) \\ &\leq r_{ZST}(BST^*(S) + b \cdot (|W| - 1)) + length(T_0) \\ &\leq r_{ZST}(BST^*(S) + 2\ length(T_0)) + length(T_0) \end{aligned}$$

and the theorem follows by observing that $length(T_0) \leq r_{SMT} BST^*(S)$ since, as noted above, the length of the minimum Steiner tree for S is a lower bound on $BST^*(S)$. □

With the currently known approximation factors for Steiner trees and zero-skew trees, Theorem 16 gives better BST approximations than Theorem 18 for the rectilinear plane, as well as arbitrary (metrically-convex) metric spaces. However, Theorem 18 may improve upon Theorem 16 for metric spaces with good Steiner tree approximation (r_{SMT} close to 1) and large Steiner ratio (r_{MST} close to 2), e.g., for high-dimensional L_p spaces.

3.7 Conclusions and open problems

We have given approximation algorithms for the ZST and BST problems with improved approximation factors for general and metrically convex metric spaces, as well as the rectilinear plane. Our algorithms have a practical running time: $O(n \log n)$ in the rectilinear plane, and $O(n^2)$ in general metric spaces. Preliminary experiments also show that, when combined with the linear time DME algorithm of [10, 13, 24], our rectilinear ZST algorithm gives results competitive to those obtained by the Greedy DME heuristic of Edahiro [25], which is regarded in the VLSI CAD community as the best ZST heuristic to date (see [50]).

An interesting open question is to determine the limitations of the spanning-tree based ZST construction introduced in this thesis. One can define the *zero-skew Steiner ratio* of a metric space as the supremum, over all sets of sinks, of the ratio between the minimum zero-skew cost (i.e., $length + delay$) of a spanning tree and the minimum ZST cost. The results in Section 3.4 imply that the zero-skew Steiner ratio is at most 4 in arbitrary metric spaces, and at most 3 in metrically convex metric spaces. On the other hand, we have constructed instances showing that the zero-skew Steiner ratio can be as large as 3 for arbitrary metric spaces; we conjecture that the ratio is never larger than 3. Determining the complexity of the zero-skew spanning tree problem is another interesting open question.

In the *planar* versions of the rectilinear ZST and BST problems, one seeks zero, respectively bounded-skew trees in the rectilinear plane with no self-intersecting edges. Charikar et al. [16] have given the first constant approximation factors for these versions; it would be interesting to find algorithms with improved approximation factors.

Chapter 4

Minimizing the number of Steiner points in bounded edge-length Steiner trees[*]

4.1 Introduction

As integrated circuit technology scales into the deep-submicron range, the effect of interconnect on chip performance becomes increasingly dominant. An important step in maintaining reasonable signal delay is to ensure that no wire segment exceeds a certain length; this can be achieved by using buffers to help interconnect global nets. Since buffers occupy a significant area on the chip and introduce additional power requirements, the goal of buffered routing is to meet the wire segment upper-bound using the minimum number of buffers. In this chapter, we concentrate on a "single-net routing" version of the problem, which is of interest when buffering is applied only to a very small number of nets. In next chapter we will give algorithms for buffered routing of a large number of nets, all competing for a limited amount of space at which buffers can be inserted.

The **Minimum number of Steiner Points Tree (MSPT) Problem** is defined as follows: given a set of terminals and a prescribed upper-bound $R > 0$, find a Steiner tree spanning the terminals and a minimum number of Steiner points such that the length of each edge in the tree is at most R. Note that, unlike the minimum length Steiner tree, the optimal MSPT may contain Steiner points of degree two. It is easy to see that the MSPT problem is equivalent to the variant in which we distinguish a source among the terminals, allow "passive" Steiner

[*]This chapter is based on joint work with Alexander Z. Zelikovsky [59].

points, i.e., branching points that do not count as buffers, and the objective is to minimize the number of buffers subject to the constraint that, on each tree path connecting the source to one of the remaining terminals, the distance between two consecutive buffers/terminals is at most R.

The MSPT problem was introduced by Sarrafzadeh and Wong [69], who considered its rectilinear and Euclidean versions, in which terminals are points in the plane and distances are measured in the L_1, respectively L_2 metrics. The rectilinear MSPT problem is of interest in VLSI, as explained above, while the Euclidean version has important applications in the design of fixed wireless networks. Unfortunately, even these restricted versions of the MSPT problem remain NP-hard [69]. While for arbitrary metric spaces the $\ln k$-approximation algorithm of Guha and Kuller [38] is best possible unless NP $\subseteq$ TIME$(n^{\log \log n})$ (cf. combined results of [50] and [27]), optimal approximation results are not yet known for the rectilinear and Euclidean planes.

Recently, Lin and Xue [55] considered the following *MST heuristic* for the MSPT problem: Compute an MST on terminals, then subdivide each edge (u, v) of the MST via $\lceil d(u, v)/R \rceil - 1$ equally spaced Steiner points, where $d(u, v)$ stands for the distance between u and v, and $R > 0$ is the prescribed edge-length upper-bound. Lin and Xue proved that the MST heuristic has an approximation factor not worse than 5 in the Euclidean plane, leaving open the problem of finding the exact approximation factor.

We give a tight analysis of the MST heuristic *for any L_p metric space*, showing that its approximation factor is exactly one less than the *MST number*, defined as the maximum possible degree of a minimum-degree MST spanning points from the space. Since the MST numbers for the rectilinear and Euclidean planes are 4 and 5 [66], our analysis implies that for these two metric spaces the MST heuristic has tight approximation factors of 3 and 4, respectively.

The factor of 4 for the Euclidean plane has been obtained independently by the authors of [17]. The analysis in [17] relies heavily on properties specific to the Euclidean plane and does not seem to extend to other metric spaces. In contrast, our analysis comes closer to the simplicity of the original argument of Lin and Xue [55], using only triangle inequality and the fact that every set of points from the space has an MST with maximum degree no larger than the MST number.

4.2 Analysis of the MST heuristic

Let (X, d) be a metric space, and let $\tau(P)$ denote the set of all d-weighted MSTs spanning $P \subseteq X$. Following Robins and Salowe [66], the *MST number* of X, $D(X)$, is defined by

$$D(X) = \sup_{P} \min_{T \in \tau(P)} \max_{v \in P} \deg_T(v), \tag{9}$$

where the supremum in (9) is taken over all finite subsets P of X. Note that, if $D(X)$ is finite, then every set of points in X admits an MST with maximum degree at most $D(X)$.

Theorem 19 *The MST heuristic has an approximation factor of $D - 1$ in every metric space whose MST number is $D < \infty$.*

Proof : Let P be a set of terminal points, and let T_{opt} be an MSPT for P. Let $s_1, \ldots, s_k$ be the Steiner points spanned by T_{opt}, numbered in the order in which a breadth-first traversal (started from an arbitrarily terminal $t_0 \in P$) encounters them. Since all edges of T_{opt} have length at most R, it follows that, for every $1 \leq i \leq k$, s_i is within a distance of R of at least one point from $P \cup \{s_1, \ldots, s_{i-1}\}$.

For a tree T, let beads$(T) = \sum_{(u,v)\in E(T)}(\lceil d(u, v)/R\rceil - 1)$ denote the number of subdivision points, or *beads*, that need to be added on T's edges in order to satisfy the edge-length condition. It is easy to see that any MST has minimum number of beads among trees spanning the same set of points; we will use this fact below.

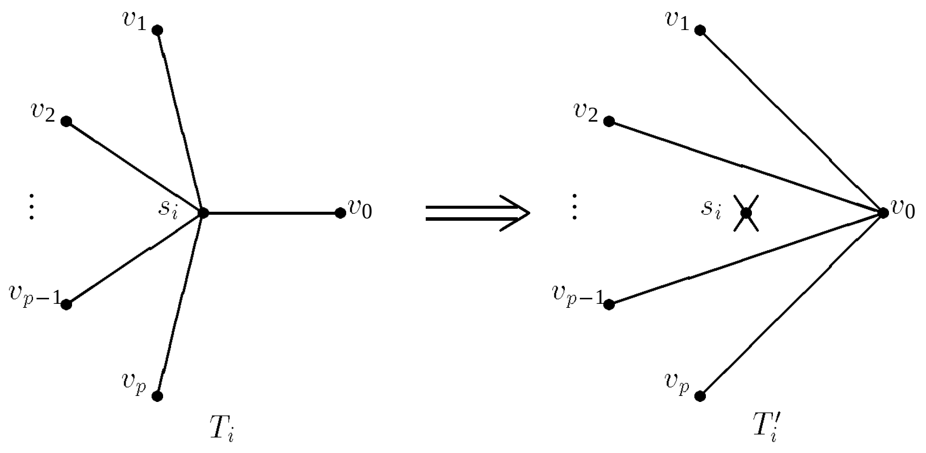

Figure 13: The basic elimination step in the proof of the MST ratio

For $1 \le i \le k$, let T_i be an MST on $P \cup \{s_1, \dots, s_i\}$ with maximum degree at most D. We claim that, for every $1 \le i \le k$,

$$\text{beads}(T_{i-1}) \le \text{beads}(T_i) + (D-1). \tag{10}$$

Let $v_0, v_1, \dots, v_p$ be the $p+1 \le D$ nodes adjacent to s_i in T_i, one of which, say v_0, must be a closest neighbor of s_i in $P \cup \{s_1, \dots, s_{i-1}\}$. Let T_i' be the tree obtained from T_i by removing s_i and connnecting to v_0 the nodes $v_i, i = 1, \dots, p$.

Note that $d(s_i, v_0) \le R$, since the BFS numbering ensures that s_i is within a distance of R of at least one point from $P \cup \{s_1, \dots, s_{i-1}\}$ and v_0 is the point from this set closest to s_i. By triangle inequality, any edge (v_j, v_0) needs at most one more bead than the edge (v_j, s_i). Hence,

$$\text{beads}(T_i') \le \text{beads}(T_i) + p \le \text{beads}(T_i) + (D-1).$$

Inequality (10) follows by noting that $\text{beads}(T_{i-1}) \le \text{beads}(T_i')$, since T_{i-1} is an MST spanning the same set of points as T_i'.

Adding inequalities (10) for $0 \leq i \leq k$ and using the fact that beads$(T_k) = 0$ gives beads$(T_0) \leq k \cdot (D-1)$. Thus, the MST on P uses at most $D-1$ times more Steiner points than T_{opt}. □

Theorem 20 *The approximation guarantee given in Theorem 19 is tight for any fixed-dimensional L_p metric space.*

Proof : Robins and Salowe [66] show that in L_p metric spaces the MST number is finite, being equal to the maximum number of points that can be placed on the surface of a unit ball such that each pair of points is strictly more than one unit apart. When the MST heuristic is run with $R = 1$ on a set of D points realizing the above configuration, the result is a tree with $D-1$ Steiner points, all of degree 2. On the other hand, the MSPT uses only one Steiner point, of degree D, namely the center of the ball. □

Since the MST number is 4 (resp. 5) for the rectilinear (resp. Euclidean) planes [66], Theorems 19 and 20 give:

Corollary 21 *The MST heuristic has a tight approximation factor of 3 in the rectilinear plane, and of 4 in the Euclidean plane.*

4.3 Conclusion and open problems

The obvious open problem is to find approximation algorithms that achieve better factors than the MST heuristic in the rectilinear plane. We believe this could be done by an adaptation of the techniques in [77, 8], based on restricted Steiner trees. Recently, [17] proposed a 3-approximation algorithm for the MSPT problem in the Euclidean plane based on these techniques.

Chapter 5

Provably good global buffering by multiterminal multicommodity flow approximation*

5.1 Introduction

Process scaling leads to an increasingly dominant effect of interconnect on high-end chip performance. Each top-level global net must undergo repeater[1] insertion to maintain signal integrity and reasonable signal delay. Estimates of the need for repeater insertion range up to 10^6 repeaters for top-level on-chip interconnect for 50nm technology. These repeaters occupy a significant area on the chip, affect global routing congestion, can entail non-standard cell height and special power routing requirements, and can act as noise sources. In a block- or reuse-based methodology, designers seek to isolate repeaters for global interconnect from individual block implementations.

For these reasons, a *buffer block* methodology has become increasingly popular in structured-custom and block-based ASIC methodologies. Two recent works by Tang and Wong [71] and Cong, Kong and Pan [20] give algorithms to solve the buffer block *planning* problem. Their formulation is roughly stated as follows: Given a placement of circuit blocks, and a set of two-pin connections with feasible regions for buffer insertion,[2] plan

*This chapter is based on joint work with F.F. Dragan, A.B. Kahng, S. Muddu, and A.Z. Zelikovsky [22, 23].

[1]Following the literature, we will use the terms *buffer* and *repeater* fairly interchangeably. When we need to be more precise: a repeater can be implemented as either an inverter or as a buffer (= two co-located inverters).

[2]In [71] only a single buffer per connection is allowed.

the location of buffer blocks within available free space so as to route maximum number of connections.

In this chapter, we address the problem of how to perform buffering of global nets *given an existing buffer block plan*. (Hence, our work is compatible with and complements the methods in [20, 71].) We give a provably good algorithm based on a recent approach of Garg and Könemann [35] and Fleischer [29]. Our method routes the nets using available buffer blocks, such that required upper and lower bounds on repeater interval—as well as length upper bounds per connection—are satisfied. In addition, our algorithm observes repeater parity constraints, i.e., it will choose to use an inverter or a buffer (= co-located pair of inverters) according to source and destination signal parity. Previous works on the problem [20, 71] assumed that global nets have been decomposed into two-pin connections. Unlike these works, our model takes into account *multiterminal nets* and allows *more than one buffer* to be inserted into any given connection.

Informally, our problem is defined as follows.

Given:

- a planar region with rectangular obstacles;
- a set of nets in the region, each net has:
  - a single source and one or more sinks;
  - a non-negative importance (criticality) coefficient;
- each sink has:
  - a parity requirement, which specifies the required parity of the number of buffers (inverters) on the path connecting it to the source;
  - a timing-driven requirement, which specifies the maximum number of buffers allowed on this path;

- a set of buffer blocks, each with given capacity; and
- an interval $[L, U]$ specifying lower and upper bounds on the distance between buffers.

The **Global Routing via Buffer Blocks (GRBB) Problem** is to route a subset of the given nets, with maximum total importance, such that:

- the distance between the source of a route and its first repeater, between any two consecutive repeaters, respectively between the last repeater on a route and the route's sink, are all between L and U;
- the number of trees passing through any given buffer block does not exceed the block's capacity;
- the number of buffers on each source-sink path should not exceed the given upper bound and should be of the given parity; to meet the parity constraint two buffers of the same block can be used.

If possible, the optimum solution to the GRBB problem simultaneously routes all the nets. Otherwise, it maximizes the sum of the importance coefficients over routed nets. The importance coefficients can be used to model various practical objectives. For example, importance coefficients of 1 for each net correspond to maximizing the number of routed nets, and importance coefficients equal the number of sinks in the net correspond to maximizing the number of connected sinks.

If all nets have exactly two terminals (the source and a single sink), the GRBB problem can be formulated as a generalized version of (vertex-capacitated) integer *multicommodity flow* (MTMCF), see [22] for details. In this chapter we show that the GRBB problem for arbitrary sized nets can be formulated as a generalized version of (vertex-capacitated) integer *multiterminal* multicommodity flow (MTMCF). Exploiting this formulation, we give a new algorithm for the GRBB problem based on randomized rounding of an approximate solution to the fractional relaxation of the integer MTMCF program. Prior to our work, multicommodity flow based heuristics have been applied [60, 70, 12, 43, 2] to unbuffered versions of VLSI global routing in which the main constraints are given by *edge*, not *vertex*, capacities. As noted in [56], the applicability of these algorithms has often been limited to problem instances of relatively small size by the prohibitive cost of solving exactly the fractional relaxation. Following [2], we avoid this limitation by using an *approximate* MTMCF algorithm. This algorithm, based on recent results of [35, 29], allows for a smooth trade-off between running time and solution accuracy. Our experiments show that even MTMCF solutions with low accuracy give good final solutions for the GRBB problem.

An interesting feature of our algorithm is its ability to work with multiterminal nets—previous work on the GRBB problem [20, 71] has considered only the case of 2-pin nets. Experiments on top-level layouts extracted from a recent high-end microprocessor design validate our MTMCF-based algorithm, and indicate that (1) the algorithm significantly outperforms existing algorithms for the problem [20], even when applied to 2-pin net decompositions, and (2) applying the MTMCF algorithm on multipin nets instead of 2-pin decompositions further increases the quality of the solution, even when the same time budget is given to both algorithms.

The rest of the chapter is organized as follows. In Section 5.2, we reduce the Global Buffering Problem to a generalized version of integer multiterminal multicommodity flow. The fractional relaxation of this problem is a special case of *packing LP*, and can thus be approximated within any desired accuracy using the algorithm of Garg and Könemann [35]. In Section 5.3 we present a faster approximation algorithm, obtained by extending the ideas of Fleischer [29] to this special type of packing LPs. In Section 5.4 we describe the randomized rounding process used to convert near-optimal fractional MCF solutions into near-optimal integral solutions. In Section 5.5 we describe a number of implemented global buffering heuristics, some based on the MTMCF approach, and some based on less sophisticated greedy ideas. Finally, in Section 5.6 we give the results of an experimental comparison of these heuristics on test cases extracted from the top-level layout of a recent high-end microprocessor, and conclude in Section 5.7 with a list of future research directions.

5.2 Integer program formulation of the GRBB problem

Given K *nets* $N_k = (s_k; t_k^1, \ldots, t_k^{q_k})$, $k = 1, \ldots, K$, and n *buffer blocks* $\{r_1, \ldots, r_n\}$, denote $S = \{s_1, \ldots, s_K\}$, $T = \{t_1^1, \ldots, t_1^{q_1}, \ldots, t_K^1, \ldots, t_K^{q_K}\}$, $R = \{r_1, \ldots, r_n\}$. Let also $c(r) \in \mathbf{N}$ denote the *capacity of the buffer block* $r \in R$, $a_k^i \in \{$*even, odd*$\}$ be the *parity requirement* for pair (s_k, t_k^i), and l_k^i be the prescribed upper bound on the number of buffers on path between source s_k and sink t_k^i.

Let p_{xy} be a rectilinear path connecting points x and y of a planar region that avoids all rectangular obstacles given in the region. Denote by $d(x, y)$ the length of a shortest such path. Let $G = (V, E)$ be a graph with vertex set $V = S \cup T \cup R$. The edge set E contains all edges of type vv, $v \in R$ (such an edge is called a loop). Two different vertices x and y are adjacent (i.e., $xy \in E$) if and only if $L \leq d(x, y) \leq U$.

A path $p = (s_k, v_1, v_2, \ldots, v_l, t_k^i)$ in G between source s_k and sink t_k^i ($k = 1, \ldots, K$, $i = 1, \ldots, q^k$) is a *restricted* (s_k, t_k^i)*-path* if

- $v_i \in R$ for each $i = 1, \ldots, l$,
- the parity of l is a_k^i,
- $l \leq l_k^i$,
- there can be some pairs of different indices $i, j \in \{1, \ldots, l\}$ such that $v_i = v_j$; in this case we must have $|i - j| = 1$.

A *feasible Steiner tree* for net N_k is a Steiner tree T_k in G connecting terminals $s_k, t_k^1, \ldots, t_k^{q_k}$ such that, for every $i = 1, \ldots, q_k$, the path of T_k connecting s_k to t_k^i is a restricted (s_k, t_k^i)-path as defined above.

Define capacities on all vertices of G by

$$c(v) := \begin{cases} 1, & \text{if } v \in S \cup T \\ \textit{capacity of buffer block } v, & \text{if } v \in R \end{cases}$$

Let $\mathcal{T}_k$ be the set of all feasible Steiner trees for net N_k, and let $\mathcal{T} = \bigcup_{k=1}^{K} \mathcal{T}_k$. For each $T \in \mathcal{T}_k$, $k = 1, \ldots, K$, define $g(T) := g_k$, where g_k is the importance of N_k.

The GRBB problem is then equivalent to the following integer linear program:

$$\begin{array}{lll} \text{maximize} & \sum_{T \in \mathcal{T}} g(T) f_T & \\ \text{s.t.} & \sum_{T \in \mathcal{T}} \pi_T(v) f_T \leq c(v), & v \in V \\ & f_T \in \{0, 1\} & T \in \mathcal{T} \end{array}$$

where $f_T = 1$ if the tree T is used in the solution and $f_T = 0$ otherwise, and $\pi_T(v)$ is the number of occurrences of v in T, i.e.,

$$\pi_T(v) := \begin{cases} 0, \text{ if } v \notin T, \\ 1, \text{ if } v \in T, \text{ but } vv \text{ is not a loop on } T, \\ 2, \text{ if } v \in T, \text{ and } vv \text{ is a loop on } T. \end{cases}$$

Our approach will be to solve the relaxation of the above integer program obtained by replacing the integrality constraint with $f_T \geq 0$ for $T \in \mathcal{T}$; we will then use randomized rounding to obtain an integer solution. We will refer to this relaxation as the *Multiterminal Multicommodity Flow* (MTMCF) LP.

Although the MTMCF LP is solvable in polynomial time (using, e.g., the ellipsoid algorithm), exact algorithms are highly impractical. On the other hand, the MTMCF LP is a special case of *packing LP,* and can thus be efficiently approximated within any desired accuracy using the recent combinatorial algorithm of Garg and Könemann [35]. In next section we give a significantly faster approximation algorithm based on a speed-up idea due to Fleischer [29]. Fleischer's idea, originally proposed for approximating the maximum edge-capacitated MCF, has been recently extended [2] to edge-capacitated *multiterminal* multicommodity flow. Here we take this approach further and show how to use it for efficient approximation of *vertex-capacitated* multiterminal multicommodity flow.

5.3 Approximation of vertex-capacitated MTMCF

Our approximation algorithm for MTMCF simultaneously solves both the primal and the dual LPs; the dual solution is used in proving the approximation guarantee of the algorithm. The dual of the MTMCF LP is:

$$\begin{array}{lll} \text{minimize} & \sum_{v\in V} w(v)c(v) & \\ \text{s.t.} & \frac{1}{g(T)}\sum_{v\in T} w(v) \geq 1, & T\in\mathcal{T} \\ & w(v)\geq 0, & v\in V \end{array}$$

The dual LP can be viewed as an assignment of non-negative weights, $w(\cdot)$, to the vertices of G such that the weight of any tree $T\in\mathcal{T}$ is at least 1; the objective is to minimize the sum $\sum_{v\in V} w(v)c(v)$. Here, the *weight, $weight(T)$, of the tree T* is the sum of the weights of vertices forming this tree (if the tree uses a loop vv then vertex v contributes twice to this sum) divided by the importance $g(T)$ of this tree.

Denote $D(w)=\sum_{v\in V} w(v)c(v)$ and let $\alpha(w)$ be the weight of a minimum weight tree from $\mathcal{T}$ (with respect to $w(\cdot)$). The dual problem is equivalent to finding a weight function $w: V\rightarrow \mathbf{R}^+$ such that $\beta=\frac{D(w)}{\alpha(w)}$ is minimized. In the following we will assume that $min\{g_k : k=1,\ldots,K\}=1$—this can be easily achieved by scaling—and will denote by Γ the maximum g_k.

In our algorithm for approximating MTMCF (Algorithm 9), $f_k(v)$ denotes how many times vertex v was used by all feasible Steiner trees found for the net N_k so far, and f denotes the total number of minimum weight feasible Steiner trees used by algorithm. The

```
Input: Graph $G$ with $K$ nets $N_1, \ldots, N_K$, vertex capacities $c(v)$
Output: Variables $f_k(v) \in [0,1]$, $k = 1, \ldots, K$, $v \in V(G)$
-----------------------------------------------------------------------
Set $f = 0$.
Set $w(v) = \delta$ for all $v \in V$.
Set $f_k(v) = 0$ for all $v \in V$ and $k = 1, \ldots, K$.
For $i = 1$ to $\log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}$ do
    For $k = 1$ to $K$ do
        Find a minimum weight tree $T$ in $\mathcal{T}_k$.
        While $weight(T) < min\{1/\Gamma, \delta/\Gamma(1+2\epsilon)^i\}$ do
            $f = f + 1$;
            For all $v \in T$, if $T$ uses a loop $vv$ then set $f_k(v) = f_k(v) + 2$ and
                $w(v) = w(v)(1 + \frac{2\epsilon}{c(v)})$; else set $f_k(v) = f_k(v) + 1$ and
                $w(v) = w(v)(1 + \frac{\epsilon}{c(v)})$.
            Find a minimum weight tree $T$ in $\mathcal{T}_k$.
        End while
    End for
End for
Output $\frac{f}{2\log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}}$, and $\frac{f_k(v)}{2\log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}}$ for each $v \in V$ and $k = 1, \ldots, K$.
```

Algorithm 9: The MTMCF approximation algorithm

algorithm associates a weight with each vertex, and every time it uses a minimum weight tree T from $\mathcal{T}_k$ ($k = 1, \ldots, K$) to connect the pins of net N_k it multiplies the weight of every vertex on this tree by $1 + \frac{\epsilon}{c(v)}$ for a fixed ϵ (if this tree uses a loop vv, then the weight of v is multiplied by $1 + \frac{2\epsilon}{c(v)}$). Initially, every vertex v has weight δ for some constant δ. Thus, the more often is a vertex used, the larger is its weight. Hence, an often used vertex is less likely to be a part of future minimum weight trees.

According to Garg and Könemann's approximation algorithm [35], we must use a lightest (with respect to current weight function $w(\cdot)$) tree from $\mathcal{T}$, if the weight of this tree is less than $1/\Gamma$. We also must stop after t iterations where t is the smallest number such that $\alpha(w)$, computed with respect to vertex weights $w(\cdot)$ of this iteration, is at least $1/\Gamma$. We extend the Fleischer's ideas [29] to our generalized fractional MTMCF problem and reduce the number of minimum weight tree computations during the algorithm. Instead of finding the lightest tree in $\mathcal{T}$ to connect the pins of a net, we settle for some tree within a factor of $(1 + 2\epsilon)$ of the lightest, and show that one can obtain a similar approximation guarantee.

Let $w_{i-1}(\cdot)$ be the weight function at the beginning of the ith iteration. We have $w_0(v) = \delta$ for each $v \in V$. For brevity denote $\alpha(w_i)$, $D(w_i)$ by $\alpha(i)$, $D(i)$ respectively. Following Fleischer, we cycle through the nets, sticking with a net until the lightest feasible Steiner tree for that net is above a $1 + 2\epsilon$ factor times a lower bound estimate of the overall lightest tree. Let $\bar{\alpha}(i)$ be a lower bound on $\alpha(i)$. To start, we set $\bar{\alpha}(0) = \delta/\Gamma$. As long as there is some $T \in \mathcal{T}$ with $weight(T) \leq min\{1/\Gamma, (1 + 2\epsilon)\bar{\alpha}(i)\}$, we use tree T. When this no longer holds, we know that the weight of the lightest tree is at least $(1 + 2\epsilon)\bar{\alpha}(i)$, and so we set $\bar{\alpha}(i + 1) = (1 + 2\epsilon)\bar{\alpha}(i)$. Thus, throughout the course of the algorithm, $\bar{\alpha}$ takes on values in the set $\{\delta/\Gamma(1 + 2\epsilon)^i\}_{i\in\mathcal{N}}$. Since $\alpha(0) \geq \delta/\Gamma$ and $\alpha(t - 1) < 1/\Gamma$, $\alpha(t) < (1+2\epsilon)/\Gamma$. Thus, when we stop, $\bar{\alpha}(t)$ is between $1/\Gamma$ and $(1+2\epsilon)/\Gamma$. Each increase of $\bar{\alpha}$ is by a $1 + 2\epsilon$ factor, hence the number of increases of $\bar{\alpha}$ is $\log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}$ (and the final value of i is $\lfloor\log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}\rfloor$).

Between updates to $\bar{\alpha}$, the algorithm proceeds by considering each net one by one. As long as the lightest feasible Steiner tree T for net N_k has weight less than the minimum of $1 + 2\epsilon$ times the current value of $\bar{\alpha}$ and $1/\Gamma$, this lightest tree T is used to connect the pins of the net N_k. When $min_{T\in\mathcal{T}_k} weight(T) \geq (1 + 2\epsilon)\bar{\alpha}$, net N_{k+1} is considered. After all K nets are considered, $\bar{\alpha}$ is updated. A total of at most $K \log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}$ minimum weight feasible Steiner tree computations are used to update α over the course of the algorithm.

Theorem 22 *Algorithm 9 is a $(1 + \omega)$-approximation algorithm for the MTMCF LP by choosing $\delta = (1 + 2\epsilon)((1 + 2\epsilon)L\Gamma)^{-\frac{1}{2\epsilon}}$ and $\epsilon < min\{.07, \frac{1+\omega-\Gamma)}{8\Gamma}\}$, where L is the number of vertices in the longest feasible Steiner tree of G connecting any net.*

Proof : Our proof is an adaptation of the proof of Garg and Könemann [35] (see also Fleischer [29]). First we show that the values $\frac{f_k(v)}{2\log_{1+2\epsilon}\frac{1+2\epsilon}{\delta}}$ $(v \in V, k = 1, \ldots, K)$, computed by the algorithm, are feasible, i.e., $\frac{1}{2\log_{1+2\epsilon}\frac{1+2\epsilon}{\delta}} \sum_{k=1}^{K} f_k(v) \leq c(v)$ and hence we do not exceed the capacity of any vertex v of G. Consider an arbitrary vertex v of G and let $M = \sum_{k=1}^{K} f_k(v)$ denotes how many times the vertex v was used by all feasible Steiner trees found by algorithm. For every two times that the vertex v was used by feasible Steiner trees, the weight of v increased by a factor of at least $(1 + \frac{2\epsilon}{c(v)})$. Since $w_0(v) = \delta$, it follows that $w_t(v) \geq \delta(1 + \frac{2\epsilon}{c(v)})^{\frac{M}{2}}$. Simplifying this expression, we get

$$w_t(v) \geq \delta(1 + \frac{2\epsilon}{c(v)})^{\frac{M}{2}} = \delta((1 + \frac{2\epsilon}{c(v)})^{c(v)})^{\frac{M}{2c(v)}} \geq \delta(1 + 2\epsilon)^{\frac{M}{2c(v)}}$$

The last time we increased the weight of v, it was on a feasible Steiner tree of weight less than $1/\Gamma$. Hence, the weight of v was less than 1. Since in each iteration we increase the vertex weight by factor of at most $(1 + 2\epsilon)$, the final weight of v is at most $(1 + 2\epsilon)$. Consequently,

$$\delta(1 + 2\epsilon)^{\frac{M}{2c(v)}} \leq w_t(v) \leq 1 + 2\epsilon, \text{ i.e., } \quad M \leq c(v) 2\log_{1+2\epsilon}\frac{1 + 2\epsilon}{\delta}$$

Now we show that the ratio of the values of the dual and the primal solutions, $\gamma = \frac{\beta}{f} 2\log_{1+2\epsilon}\frac{1+2\epsilon}{\delta}$, is at most $(1 + \omega)$.

For each iteration $i \geq 1$ we have

$$\begin{aligned} D(i) &= \sum_{v \in V} w_i(v)c(v) = \sum_{v \in V} w_{i-1}(v)c(v) + \epsilon \sum_{v \in T} w_{i-1}(v) \\ &\leq D(i-1) + \epsilon(1 + 2\epsilon)\Gamma\alpha(i-1) \end{aligned}$$

Note that, if T used a loop vv, then v contributes to the sum $\sum_{v \in T} w_{i-1}(v)$ twice (since $w_i(v) = w_{i-1}(v)(1 + \frac{2\epsilon}{c(v)})$).

Then,

$$D(i) - D(0) \leq \epsilon(1 + 2\epsilon)\Gamma \sum_{j=1}^{i} \alpha(j-1)$$

Consider the weight function $w_i(\cdot) - w_0(\cdot)$. We have $\alpha(w_i - w_0) \geq \alpha(w_i) - \delta L$, where L is the number of vertices in the longest feasible Steiner tree of G connecting any net.

Consequently, if $\alpha(w_i) - \delta L > 0$, then

$$\beta \leq \frac{D(w_i - w_0)}{\alpha(w_i - w_0)} \leq \frac{D(i) - D(0)}{\alpha(i) - \delta L} \leq \frac{\epsilon(1+2\epsilon)\Gamma \sum_{j=1}^{i} \alpha(j-1)}{\alpha(i) - \delta L}$$

Thus, in any case (for the case $\alpha(w_i) - \delta L \leq 0$, it is trivial) we have

$$\begin{aligned}
\alpha(i) &\leq \delta L + \frac{\epsilon(1+2\epsilon)\Gamma}{\beta} \sum_{j=1}^{i} \alpha(j-1) \\
&\leq (1 + \frac{\epsilon(1+2\epsilon)\Gamma}{\beta})^{i-1}(\delta L + \frac{\epsilon(1+2\epsilon)\Gamma}{\beta}\alpha(0)) \\
&\leq (1 + \frac{\epsilon(1+2\epsilon)\Gamma}{\beta})^{i-1}(\delta L + \frac{\epsilon(1+2\epsilon)\Gamma}{\beta}\delta L) \\
&= \delta L(1 + \frac{\epsilon(1+2\epsilon)\Gamma}{\beta})^{i} \leq \delta L e^{\frac{i\epsilon(1+2\epsilon)\Gamma}{\beta}}
\end{aligned}$$

For the last inequality the fact $1 + x \leq e^x$ for $x \geq 0$ is used.

Since we stop at iteration t with $\alpha(t) \geq 1/\Gamma$, and $t = f$, we get

$$1/\Gamma \leq \alpha(t) \leq \delta L e^{\frac{t\epsilon(1+2\epsilon)\Gamma}{\beta}} = \delta L e^{\frac{f\epsilon(1+2\epsilon)\Gamma}{\beta}}.$$

Hence,

$$\frac{\beta}{f} \leq \frac{\epsilon(1+2\epsilon)\Gamma}{\ln(\delta L \Gamma)^{-1}}$$

Now, for the ratio γ we obtain

$$\begin{aligned}
\gamma &= \frac{\beta}{f} 2 \log_{1+2\epsilon} \frac{1+2\epsilon}{\delta} \\
&\leq \frac{2\epsilon(1+2\epsilon)\Gamma \log_{1+2\epsilon} \frac{1+2\epsilon}{\delta}}{\ln(\delta L \Gamma)^{-1}} = \frac{2\epsilon(1+2\epsilon)\Gamma \ln \frac{1+2\epsilon}{\delta}}{\ln(1+2\epsilon)\ln(\delta L \Gamma)^{-1}}
\end{aligned}$$

Since we have chosen $\delta = (1+2\epsilon)((1+2\epsilon)L\Gamma)^{\frac{-1}{2\epsilon}}$, we get

$$\frac{\ln\frac{1+2\epsilon}{\delta}}{\ln(\delta L\Gamma)^{-1}} = \frac{1}{1-2\epsilon}$$

and hence,

$$\gamma \leq \frac{2\epsilon(1+2\epsilon)\Gamma}{(1-2\epsilon)\ln(1+2\epsilon)} \leq \frac{2\epsilon(1+2\epsilon)\Gamma}{(1-2\epsilon)(2\epsilon-4\epsilon^2/2)}$$
$$\leq (1+2\epsilon)(1-2\epsilon)^{-2}\Gamma$$

Here we use that $\ln(1+x) \geq x - x^2/2$ (by Taylor series expansion of $\ln(1+x)$).

Since $(1+2\epsilon)(1-2\epsilon)^{-2}$ is at most $(1+8\epsilon)$, for $\epsilon < .07$, and $(1+8\epsilon)\Gamma$ should be no more than our approximation ratio $(1+\omega)$, we are done. $\square$

In Algorithm 9 we need to solve the following problem. Let G_k ($k = 1, \ldots, K$) be a subgraph of the graph G induced by vertices $\{s_k, t_k^1, \ldots, t_k^{q_k}\} \cup R$ (recall that each vertex $v \in R$ has a loop $vv \in E$). Let also each vertex v of G_k have a non-negative weight $w(v)$. Find a minimum weight tree T_k in G_k connecting s_k with $t_k^1, \ldots, t_k^{q_k}$ such that, for each $i = 1, \ldots, q_k$, the path of T_k between s_k and t_k^i passes through even (odd, depending on a_k^i) number of vertices, and that number of vertices should not exceed l_k^i. This path may contain a loop. So, the vertex weight will contribute either once or twice (in case of loop) to the weight of the tree T_k.

Let $L_k = max\{l_k^1, \ldots, l_k^{q_k}\}$. We reduce this problem to the usual shortest directed rooted Steiner tree problem on an edge-weighted directed acyclic graph (dag) D_k with $V(D_k) = \{s_k\} \cup \{r_{i,j} \mid 1 \leq i \leq n, 1 \leq j \leq L_k\} \cup \{t_k^1, \ldots, t_k^{q_k}\}$ and $E(D_k) = E_1 \cup E_2 \cup E_3$, where

$$E_1 = \{(s_k, r_{i,1}) \mid 1 \leq i \leq n, (s_k, r_i) \in E(G)\}$$
$$E_2 = \{(r_{i,j}, r_{i',j+1}) \mid 1 \leq i, i' \leq n, 1 \leq j < L_k, (r_i, r_{i'}) \in E(G)\}$$
$$E_3 = \{(r_{i,j}, t_k^h) \mid 1 \leq i \leq n, 1 \leq h \leq q_k, 1 \leq j \leq l_k^h, j \equiv a_k^h (\text{mod } 2), (r_i, t_k^h) \in E(G)\}$$

If the cost of each arc (x, y) in D_k is given by $w(x)$, it is easy to see that finding the minimum weight tree in $\mathcal{T}_k$ reduces to finding a minimum cost directed rooted Steiner tree (DRST) in D_k. Generally, the *directed rooted Steiner tree problem* asks, for a given directed edge-weighted graph $H = (X, U)$, a specified root $r \in X$, and a set of terminals $Y \subset X$, to find the minimum cost arborescence rooted at r and spanning all the vertices in Y (in other words r should have a path to every vertex in Y). Unfortunately, the fact that D_k is acyclic does not help. There is a simple reduction for this problem from arbitrary directed graphs to acyclic graphs. As far as we know, the best result for the DRST problem is due to Charikar et al. [15] which says that an $O(\log^2 q_k)$-approximate solution can be found in quasi-polynomial time $O(n^{3 \log q_k})$. Since this is very inefficient, we need to find some other ways to compute such trees. One way is to compute (exactly or approximately) a DRST once, and then in all next iterations (with new edgelengths) using the found Steiner points $p_1, \ldots, p_s$ find a minimum directed spanning tree of the graph induced by $\{s_k, t_k^1, \ldots, t_k^{q_k}, p_1, ..., p_s\}$ (this approach was used in [2]). To find a minimum spanning directed tree in directed acyclic graphs, one can use a very simple procedure: for each vertex choose a shortest incoming arc. After running this procedure one can recursively delete all leaves of the spanning tree, that are not sinks of the net N_k.

5.4 Rounding the fractional MTMCF

In the previous section we presented an algorithm for approximating the optimum multiterminal multicommodity flow (MTMCF) within any desired accuracy. The optimum MTMCF gives an upper-bound on the maximum number of routable nets (connections). In this section we show how to use the approximate MTMCF to route an almost optimal number of nets (resp. connections). Our construction is based on the randomized rounding technique of Raghavan and Thomson [63], in particular, on the random-walk based algorithm for rounding multicommodity flow [62] (see also [56]).

Input: Multiterminal flows $f_k(e) \in [0,1]$, $k = 1, \ldots, K$, $e \in E(G)$
Output: Set of trees $T_k \in \mathcal{T}_k$

```
For each $k = 1, \ldots, K$, with probability $f_k$, do
    $T_k \leftarrow \{s_k\}$
    For each sink $t_k^i$ in $N_k$ do
        $P \leftarrow \emptyset$;    $v \leftarrow t_k^i$
        While $v \notin T_k$ do
            Pick arc $(u,v)$ with probability $f_k(u,v)/f_k(v)$
            $P \leftarrow P \cup \{(u,v)\}$;    $v \leftarrow u$
        End while
        $T_k \leftarrow T_k \cup P$
    End for
End for
```

Algorithm 10: The randomized MTMCF rounding algorithm

The MCF rounding algorithm in [62] chooses a set of source-sink pairs by including each pair (s,t) with a probability equal to the flow from s to t. Then, for each chosen pair, (s,t), the algorithm performs a random-walk from s to t, based on probabilities given by edge-flows. In our MTMCF rounding algorithm (Algorithm 10), a net $N_k = (s_k; t_k^1, \ldots, t_k^{q_k})$ is also routed with probability equal to the net's total flow, $f_k = \sum_{T \in \mathcal{T}_k} f_T$. Since we need to construct a tree connecting all sinks t_k^i to the source s_k, we route the net by performing *backward* random walks from each sink until reaching either s_k or a vertex on a path already included in the tree. Thus, if the net has only one sink, our rounding algorithm becomes identical to the algorithm in [62], except for the direction of the random walk.

Ensuring that no vertex capacities are exceeded can be accomplished in two ways. Following [56], one way is to solve the MTMCF LP with capacities scaled down by a small factor that guarantees that the rounded solution will meet the *original* capacities with very high probability. A simpler approach, the so-called *greedy-deletion algorithm* [22], is to repeatedly drop routed nets that visit over-used vertices until feasibility is achieved. We implement a modification of the second approach: instead of dropping an entire tree, we drop only the sinks which use paths through over-used vertices.

5.5 Implemented algorithms

In this section we describe the implemented algorithms for the Global Routing via Buffer Blocks problem.

5.5.1 Greedy routing algorithms

We have implemented 3 greedy algorithms for the GRBB problem. The first algorithm, similar to one proposed in [20], starts by decomposing each multiterminal net into 2-terminal nets. Then, the algorithm attempts to route the 2-terminal nets one by one, using for routing a shortest available path from the net's source to its sink, if such a path exists. We will refer to this algorithm as the *forward 2-terminal greedy* (F-2TG) algorithm.

The second greedy algorithm (Algorithm 11), referred to as the *multiterminal Greedy* (MTG) algorithm, handles multiterminal nets, and thus does not have to resort to net decomposition. For each multiterminal net we attempt to route the sinks one by one. For each sink we use a shortest available path to one of the vertices already connected to the source, if any such path exists.

The third algorithm, the *backward 2-terminal greedy* (B-2TG), works as F-2TG, except for the fact that shortest paths are computed backward, from sinks toward sources and not from sources toward sinks. Notice that B-2TG is the special case of MTG when applied to 2-terminal nets.

```
Input: Graph $G$ with $K$ nets $N_1, \dots, N_K$, vertex capacities $c(v)$
Output: Set of trees $T_k \in \mathcal{T}_k$
--------------------------------------------------------------------------
For each $k = 1, \dots, K$, do
    $T_k \leftarrow \{s_k\}$
    For each sink $t_k^i$ in $N_k$ do
        Using a backward BFS search, find a shortest path $P$ from $t_k^i$ to $T_k$ in $G$
          using only vertices $v$ with $c(v) > 0$; if no such path exists let $P = \emptyset$
        $T_k \leftarrow T_k \cup P$
        For each vertex $v$ in $P$, $c(v) \leftarrow c(v) - 1$
    End for
End for
```

Algorithm 11: The multiterminal greedy (MTG) routing algorithm

5.5.2 Flow rounding algorithms

We have implemented two flow rounding algorithms. The first algorithm (Algorithm 12) is based on MTMCF rounding. Our current implementation decomposes larger nets into 3-terminal nets before applying the MTMCF routing algorithm, we will refer to this implementation as 3TMCF. For 3-terminal nets we can find the optimum directed routed Steiner tree efficiently, and we do not need to resort to the approximations suggested at the end of Section 5.3.

In order to assess the benefit of using multiterminal nets, we include in our comparison a second flow rounding algorithm, which starts by decomposing each multiterminal net into 2-terminal nets and then solves approximately the fractional relaxation of the resulting integer multicommodity flow program and applies randomized rounding. This algorithm will be referred to as 2TMCF.

```
Input: Graph $G$ with $K$ nets $N_1, \ldots, N_K$, vertex capacities $c(v)$
Output: Set of trees $T_k \in \mathcal{T}_k$
------------------------------------------------------------
Find an approximate MTMCF using Algorithm 9.
Round the approximate MTMCF using Algorithm 10.
Use greedy deletion to find a feasible integer solution.
Use the MTG Algorithm 11 on the unrouted nets to find a maximal routing.
```

Algorithm 12: The MTMCF routing algorithm

5.6 Implementation experience

All experiments were conducted on a SGI Origin 2000 with 16 195MHz MIPS R10000 processors—only one of which is actually used by the sequential implementations included in our comparison—and 4 G-Bytes of internal memory, running under IRIX 6.4 IP27. Timing was performed using low-level Unix interval timers, under similar load conditions for all experiments. All algorithms were coded in C and compiled using gcc version egcs-2.91.66 with -O4 optimization.

The three test cases used in our experiments were extracted from the next-generation microprocessor chip at SGI. We used an optimized floorplan of the circuit blocks and also optimized the location of the source/sink pin locations based on coarse timing budgets. We used $U = 4000\mu$m, and varied L between 500μm and 2000μm. Path-length upper-bounds were computed with the formula $l_k = $ dist(s_k, t_k)/1000. In all test cases considered the number of nets was large (over 6000), and the number of buffer blocks small (50), with relatively large capacity (400 buffers per block); such values are typical for this application [22].

Tables 4–6 give the number of routed sinks and the running time on the three instances by each of the algorithms included in our comparison. Figure 5.6 plots, for one of the instances, the solution quality versus CPU time (in seconds, excluding I/O and memory allocation) for each of the considered algorithms.

Instance				GREEDY		
ID	Nets	Sinks	N/S	F-2TG	B-2TG	MTG
i1	4764	6038	2.27	**89.5** *0.58*	**90.6** *0.54*	**93.5** *0.53*
i2	4925	6296	2.28	**89.9** *0.84*	**91.6** *0.58*	**93.6** *0.55*
i3	4938	6321	2.28	**89.8** *0.65*	**91.5** *0.59*	**93.3** *0.54*

Table 4: Percent of sinks connected and CPU time for the greedy algorithms

Instance				2TMCF					
ID	Nets	Sinks	N/S	$\varepsilon = 0.64$	$\varepsilon = 0.32$	$\varepsilon = 0.16$	$\varepsilon = 0.08$	$\varepsilon = 0.04$	$\varepsilon = 0.02$
i1	4764	6038	2.27	**94.8** *2.84*	**95.8** *12.13*	**96.5** *39.50*	**96.6** *139.83*	**96.8** *600.89*	**96.8** *2321.67*
i2	4925	6296	2.28	**96.2** *4.35*	**97.1** *11.34*	**97.4** *40.55*	**97.5** *156.89*	**97.6** *690.31*	**97.6** *2604.34*
i3	4938	6321	2.28	**96.2** *3.37*	**96.9** *11.08*	**97.3** *40.84*	**97.3** *163.32*	**97.5** *730.95*	**97.5** *2638.04*

Table 5: Percent of sinks connected and CPU time for the 2TMCF algorithm

Instance				3TMCF					
ID	Nets	Sinks	N/S	$\varepsilon = 0.64$	$\varepsilon = 0.32$	$\varepsilon = 0.16$	$\varepsilon = 0.08$	$\varepsilon = 0.04$	$\varepsilon = 0.02$
i1	4764	6038	2.27	**95.7** *16.57*	**96.8** *53.62*	**97.3** *203.03*	**97.5** *817.59*	**97.6** *3166.03*	**97.6** *12736.22*
i2	4925	6296	2.28	**97.0** *19.50*	**98.0** *64.13*	**98.4** *242.17*	**98.5** *942.34*	**98.6** *3721.95*	**98.4** *14854.06*
i3	4938	6321	2.28	**96.8** *18.99*	**97.8** *66.12*	**98.3** *246.29*	**98.4** *956.83*	**98.4** *3813.42*	**98.3** *15088.50*

Table 6: Percent of sinks connected and CPU time for the 3TMCF algorithm

The first surprising thing to notice is that B-2TG gives noticeably better results than F-2TG, despite the fact that the two algorithms are nearly identical (they both add paths of the same length until some of the vertices use up the full capacity).[3] Perhaps not so surprising is the fact that the multiterminal greedy algorithm is better than both F-2TG and B-2TG. Notice that the running time of all three greedy algorithms is virtually the same, so MTG is the clear choice among them.

Our experiments clearly demonstrate the high quality of the solutions obtained by flow rounding methods. Significant improvement over the best of the greedy methods is possible even with a very small increase in running time, proof that even very coarse MCF/MTMCF approximations give helpful hints to the randomized rounding procedure. Since randomized rounding is very fast, faster in fact than any of the greedy algorithms, the MCF/MTMCF algorithms can be further improved by running randomized rounding with the same fractional flow a large number of times and taking the best of the rounded solutions; our current implementation does not exploit this idea.

Finally, our experiments show that even a limited use of multiterminal nets (decomposition into nets of size 3) gives improvements over the already very high-quality MCF algorithm of Dragan et al. [22]. In fact, the 3TMCF algorithm outperforms the MCF algorithm in [22] even when the same time budget is given to both algorithms.

5.7 Conclusions and future research directions

In this chapter, we addressed the problem of how to perform buffering of global nets given an existing buffer block plan. We gave a provably good algorithm based on a novel approach to MTMCF approximation inspired by recent results of Garg and Könemann [35]

[3]The advantage in computing backward shortest paths, as opposed to forward shortest paths, appears to lie in the fact that the former gives a set of paths that are better spread out in the vicinity of the source of a large net. If the sinks of such a net are grouped in a small number of clusters, which is typically the case in real designs, the forward greedy algorithm is likely to use a small number of neighbors of the source for all these paths, thus leading to the faster exhaustion of the available capacity in these vertices.

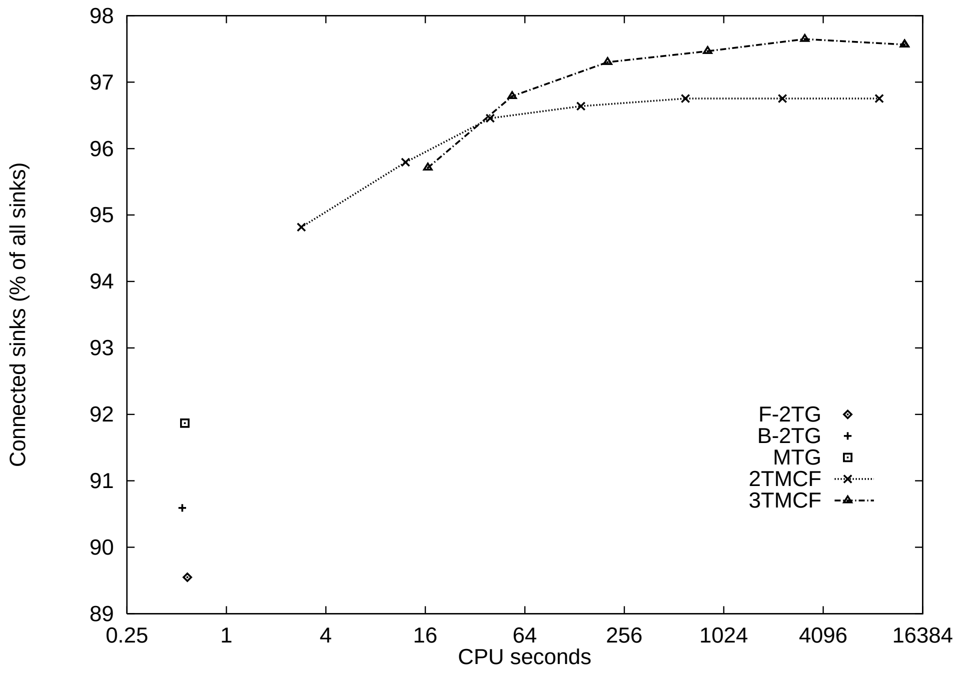

Figure 14: Percent of sinks connected vs. time on instance i1

and Fleischer [29] on edge-capacitated MCF. Our MTMCF algorithm outperforms existing algorithms for the problem [20], and has been validated on top-level layouts extracted from a recent high-end microprocessor design.

As presented here, our work targets the very early global wireplanning activities, i.e., pre-synthesis chip planning. It should be interesting to extend the class of methodologies to which the MTMCF approach applies. Possible directions in which the approach can be extended are: (1) handling routing congestion, e.g., by introducing capacitated “virtual” nodes in the flow graph; (2)handling timing criticality and budgets, e.g., by better use of net ordering and weighting; (3)improved decomposition heuristics, perhaps based on clustering techniques; and (4) more accurate treatement of multiterminal nets, by decomposition into nets with more than 3 terminals.

Vita

Ion I. Măndoiu was born on November 30, 1967 in Strejeşti, a little town on the right shore of the Olt river in Romania. Between 1982 and 1986 he attended the "Ion Minulescu" high school in the nearby city of Slatina. Upon high school graduation he was admitted to the University of Bucharest, which he started to attend in 1987 after nine months of military service. In June 1992 he graduated, first in his class, with a M.S. in Computer Science. He has been with the Department of Computer Science of the University of Bucharest since Octomber 1992, on leave since 1995 when he entered the Ph.D. program in the College of Computing at Georgia Institute of Technology. His research interests include the design, analysis, and implementation of exact and approximation algorithms, combinatorial optimization, algorithmic problems in coding and information theory, and VLSI computer-aided design.

Bibliography

[1] A. Agrawal, P. Klein, and R. Ravi. When trees collide: an approximation algorithm for the generalized Steiner problem in networks. *SIAM J. Comput.*, 24:440–456, 1995.

[2] Ch. Albrecht. Provably good global routing by a new approximation algorithm for multicommodity flow. In *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2000.

[3] S. Arora. Polynomial time approximation scheme for Euclidean TSP and other geometric problems. In *Proc. 37th IEEE Ann. Symp. Found. Comput. Sci.*, pages 2–11, 1996.

[4] S. Arora. Nearly linear time approximation scheme for Euclidean TSP and other geometric problems. In *Proc. 38th IEEE Ann. Symp. Found. Comput. Sci.*, pages 554–563, 1997.

[5] H. Bakoglu. *Circuits, interconnections, and packaging for VLSI*. Addison-Wesley, Reading, Massachusetts, 1990.

[6] H. Bakoglu, J.T. Walker, and J.D. Meindl. A symmetric clock-distribution tree and optimized high-speed interconnections for reduced clock-skew in ULSI and WSI circuits. In *Proc. IEEE Int. Conf. Comput. Design*, pages 118–122, 1986.

[7] P. Berman, U. Fössmeier, M. Karpinski, M. Kaufmann, and A. Zelikovsky. Approaching the $5/4$ approximation for rectilinear Steiner trees. Technical Report WSI-94-06, Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Tübingen, Germany, 1994.

[8] P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. *J. Algorithms*, 17:381–408, 1994.

[9] S.N. Bespamyatnikh. An optimal algorithm for closest-pair maintenance. *Discret. Comput. Geometry*, 19:175–195, 1998.

[10] K.D. Boese and A.B. Kahng. Zero-skew clock routing trees with minimum wirelength. In *Proc. IEEE Int. ASIC Conf.*, pages 17–21, 1992.

[11] M. Borah, R. M. Owens, and M. J. Irwin. A fast and simple Steiner routing heuristic. *Discret. Appl. Math.*, 90:51–67, 1999.

[12] R. C. Carden and C.-K. Cheng. A global router using an efficient approximate multicommodity multiterminal flow algorithm. In *Proc. ACM/IEEE Design Automation Conf.*, pages 316–321, 1991.

[13] T.-H. Chao, Y.-C. Hsu, and J.-M. Ho. Zero skew clock net routing. In *Proc. ACM/IEEE Design Automation Conf.*, pages 518–523, 1992.

[14] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, and A. B. Kahng. Zero skew clock routing with minimum wirelength. *IEEE Trans. Circ. and Syst. II: Analog & Digital Sign. Process.*, 39:799–814, 1992.

[15] M. Charikar, Ch. Chekuri, T. Cheung, Z. Dai, A. Goel, and S. Cheung. Approximation algorithms for directed steiner problems. *J. Algorithms*, 33:73–91, 1999.

[16] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai, and A. Tomkins. Minimizing wirelength in zero and bounded skew clock trees. In *Proc. 10th ACM-SIAM Ann. Symp. Discret. Algorithms*, pages 177–184, 1999.

[17] D. Chen, D.-Z. Du, X.-D. Hu, G.-H. Lin, L. Wang, and G. Xue. Approximations for Steiner trees with minimum number of Steiner points. *manuscript*, 1999.

[18] J. Cong, A. Kahng, and G. Robins. Matching-based methods for high-performance clock routing. *IEEE Trans. CAD*, 12:1157–1169, 1993.

[19] J. Cong, A.B. Kahng, C.K. Koh, and C.-W. Tsao. Bounded-skew clock and Steiner routing. *ACM Trans. Design Automation*, 3:341–388, 1998.

[20] J. Cong, T. Kong, and D. Z. Pan. Buffer block planning for interconnect-driven floorplanning. In *Proc. IEEE Int. Conf. on CAD,* pages 358–363, 1999.

[21] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *Introduction to algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[22] F.F. Dragan, A.B. Kahng, I.I. Măndoiu, S. Muddu, and A.Z. Zelikovsky. Provably good global buffering using an available buffer block plan. In *Proc. IEEE Int. Conf. on CAD*, 2000 (to appear).

[23] F.F. Dragan, A.B. Kahng, I.I. Măndoiu, S. Muddu, and A.Z. Zelikovsky. Provably good global buffering by multiterminal multicommodity flow approximation. Submitted.

[24] M. Edahiro. Minimum skew and minimum path length routing in VLSI layout design. *NEC Research and Development*, 32:569–575, 1991.

[25] M. Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *Proc. 30th ACM/IEEE Design Automation Conf.*, pages 612–616, 1993.

[26] D. Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *ACM J. Experimental Algorithmics*, 5:1–23, 2000.

[27] U. Feige. A treshold of $\ln n$ for approximating set cover. In *Proc. 28th ACM Symp. Theor. Comput.*, pages 314–318, 1996.

[28] U. Feige. A treshold of $\ln n$ for approximating set cover. *J. ACM*, 45:634–652, 1998.

[29] L. Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. In *Proc. 40th IEEE Ann. Symp. Found. Comput. Sci.*, pages 24–31, 1999.

[30] U. Fößmeier and M. Kaufmann. Solving rectilinear Steiner tree problems exactly in theory and practice. In *Proc. 5th Eur. Symp. Algorithms*, volume 1284 of *Lecture Notes in Computer Science*, Berlin, Germany, 1997. Springer-Verlag.

[31] U. Fößmeier, M. Kaufmann, and A. Z. Zelikovsky. Faster approximation algorithms for the rectilinear Steiner tree problem. *Discret. Comput. Geometry*, 18:93–109, 1997.

[32] J. L. Ganley. Computing optimal rectilinear Steiner trees: A survey and experimental evaluation. *Discret. Appl. Math.*, 89:161–171, 1998.

[33] J. L. Ganley and J. P. Cohoon. Improved computation of optimal rectilinear Steiner minimal trees. *Int. J. Comput. Geometry and Appl.*, 7:457–472, 1997.

[34] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32:826–834, 1977.

[35] N. Garg and J. Könemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In *Proc. 39th IEEE Ann. Symp. Found. Comput. Sci.*, pages 300–309, 1998.

[36] M.X. Goemans and D.P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995.

[37] J. Griffith, G. Robins, J.S. Salowe, and T. Zhang. Closing the gap: near-optimal Steiner trees in polynomial time. *IEEE Trans. CAD*, 13:1351–1365, 1994.

[38] S. Guha and S. Khuller. Improved methods for approximating node weighted Steiner trees and connected dominating sets. *Inf. Comput.*, 150:57–74, 1999.

[39] R.-H. Güting, O. Nurmi, and T. Ottmann. Fast algorithms for direct enclosures and direct dominances. *J. Algorithms*, 10:170–186, 1989.

[40] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM J. Appl. Math.*, 14:255–265, 1966.

[41] J. Ho, G. Vijayan, and C. K. Wong. New algorithms for the rectilinear Steiner tree problem. *IEEE Trans. CAD*, 9:185–193, 1990.

[42] M. Hollander and D.A. Wolfe. *Nonparametric statistical methods*. John Wiley & Sons, 2nd edition, 1999.

[43] J. Huang, X.-L. Hong, C.-K. Cheng, and E.S. Kuh. An efficient timing-driven global routing algorithm. In *Proc. ACM/IEEE Design Automation Conf.*, pages 595–600, 1993.

[44] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM J. Appl. Math.*, 30:104–114, 1976.

[45] F. K. Hwang. An $O(n \log n)$ algorithm for rectilinear minimal spanning trees. *J. ACM*, 26:177–182, 1979.

[46] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner tree problem*, volume 53 of *Annals of Discrete Mathematics*. North-Holland, Amsterdam, Netherlands, 1992.

[47] M.A.B. Jackson, A. Srinivasan, and E.S. Kuh. Clock routing for high-performance ICs. In *Proc. ACM/IEEE Design Automation Conf.*, pages 574–579, 1990.

[48] A. B. Kahng, J. Cong, and G. Robins. High-performance clock routing based on recursive geometric matching. In *Proc. ACM/IEEE Design Automation Conf.*, pages 574–579, 1990.

[49] A. B. Kahng and G. Robins. A new class of iterative Steiner tree heuristics with good performance. *IEEE Trans. CAD*, 11:893–902, 1992.

[50] A. B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*. Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[51] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted Steiner trees. *J. Algorithms*, 19:104–115, 1995.

[52] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for Steiner trees. *Acta Inform.*, 15:141–145, 1981.

[53] F. D. Lewis, W. C.-C. Pong, and N. Van Cleave. Local improvement in Steiner trees. In *Proc. 3rd Great Lakes Symp. VLSI*, pages 105–106, 1993.

[54] Y.M. Li and M.A. Jabri. A zero-skew clock routing scheme for VLSI circuits. In *Proc. IEEE Int. Conf. on CAD*, pages 458–463, 1992.

[55] G.-H. Lin and G. Xue. Steiner tree problem with minimum number of Steiner points and bounded edge-length. *Inf. Process. Lett.*, 69:53–57, 1999.

[56] R. Motwani, J. Naor, and P. Raghavan. Randomized approximation algorithms in combinatorial optimization. In D.S. Hochbaum, editor, *Approximation algorithms for NP-hard problems*, pages 144–191, Boston, MA, 1997. PWS Publishing.

[57] I. I. Măndoiu, V.V. Vazirani, and J.L. Ganley. A new heuristic for rectilinear Steiner trees. In *Proc. IEEE Int. Conf. on CAD*, pages 157–162, 1999.

[58] I. I. Măndoiu, V.V. Vazirani, and J.L. Ganley. A new heuristic for rectilinear Steiner trees. *IEEE Trans. CAD*, 19, 2000 (to appear).

[59] I. I. Măndoiu and A.Z. Zelikovsky. A note on the MST heuristic for bounded edge-length Steiner trees with minimum number of Steiner points. *Information Processing Letters*, 75:165–167, to appear Oct. 2000.

[60] A.P.-C Ng, P. Raghavan, and C.D. Thomson. Experimental results for a linear program global router. *Computers and Artificial Intelligence*, pages 229–242, 1987.

[61] H. J. Prömel and A. Steger. RNC-approximation algorithms for the Steiner problem. In R. Reischuk and M. Morvan, editors, *Proc. Symp.Theor. Aspect. Comput. Sci.*, volume 1200 of *Lecture Notes in Computer Science*, pages 559–570. Springer-Verlag, Berlin, Germany, 1997.

[62] P. Raghavan and C.D. Thomson. Provably good routing in graphs: Regular arrays. In *Proc. 17th ACM Symp. Theor. Comput.*, pages 79–87, 1985.

[63] P. Raghavan and C.D. Thomson. Randomized rounding. *Combinatorica*, pages 365–374, 1987.

[64] S. Rajagopalan and V.V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM J. Comput.*, 28:526–541, 1999.

[65] G. Robins. Batched Iterated 1-Steiner code available at www.cs.virginia.edu/~robins/steiner.tar.

[66] G. Robins and J. S. Salowe. Low-degree minimum spanning trees. *Discret. Comput. Geometry*, 14:151–165, 1995.

[67] G. Robins and A. Zelikovsky. Improved Steiner tree approximation in graphs. In *Proc. 11th ACM-SIAM Ann. Symp. Discret. Algorithms*, pages 770–779, 2000.

[68] J. S. Salowe and D. M. Warme. Thirty-five-point rectilinear Steiner minimal trees in a day. *Networks*, 25:69–87, 1995.

[69] M. Sarrafzadeh and C. K. Wong. Bottleneck Steiner trees in the plane. *IEEE Trans. Comput.*, 41:370–374, 1992.

[70] E. Shragowitz and S. Keel. A global router based on a multicommodity flow model. *Integration*, 5:3–16, 1987.

[71] X. Tang and D. F. Wong. Planning buffer locations by network flows. In *Proc. ACM/SIGDA Int. Symp. Phys. Design*, 2000.

[72] V.V. Vazirani. *Approximation algorithms*. Springer-Verlag, Berlin, 2000.

[73] D. M. Warme, P. Winter, and M. Zacharisen. Exact algorithms for plane Steiner tree problems: A computational study. Technical Report DIKU-TR-98/11, Department of Computer Science, University of Copenhagen, Copenhagen, Denmark, 1998.

[74] D.M. Warme, P. Winter, and M. Zacharisen. The GeoSteiner 3.0 package, available at ftp.diku.dk/diku/users/martinz/geosteiner-3.0.tar.gz.

[75] Y. F. Wu, P. Widmayer, and C. K. Wong. A faster approximation algorithm for the Steiner problem in graphs. *Algorithmica*, 23:223–229, 1986.

[76] M. Zachariasen. Rectilinear full Steiner tree generation. *Networks*, 33:125–143, 1999.

[77] A. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.

[78] A.Z. Zelikovsky and I. I. Măndoiu. Practical approximation algorithms for zero- and bounded-skew clock trees. Submitted.