Advanced Interior Point Formulation for the Global Routing Problem

by

David Chor-Cheung Wong

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2009

Abstract

As the circuit size increases in modern electronics, the design process becomes more complicated. Even though the hardware design process is divided into multiple phases, many of the divided problems are still extremely time consuming to solve. One of these NP-hard problems is the routing problem. As electronics step into the deep submicron era, optimizing the routing becomes increasingly important.

One of the methods to solve global routing is to formulate the problem as an integer programming (IP) problem. This formulation can then be relaxed into a linear programming problem and solved using interior point method. This thesis investigates two new approaches to optimize the speed of solving global routing using Karmarkar's interior point method, as well as the effect of combining various optimizations with these new approaches. The first proposed approach is to utilize solution stability as the interior point loop converges, and attempt to remove solutions that have already stabilized. This approach reduces the problem size and allows subsequent interior point iterations to proceed faster. The second proposed approach is to solve the inner linear system (projection step) in interior point method in parallel.

Experimental results show that for large routing problems, the performance of the solver is improved by the optimization approaches. The problem reduction stage allows for great speedup in the interior point iterations, without affecting the quality of the solution significantly. Furthermore, the timing required to solve inner linear system in the interior point method is improved by solving the problem in parallel. With these optimizations, solving the routing problem using the IP formation becomes increasingly more efficient. By solving an efficient parallel IP formation rather than a traditional sequential approach, more efficient optimal solutions which incorporate multiple conflicting objectives can be achieved.

Acknowledgements

I am greatly thankful for all the assistance, guidance, and unlimited support from my supervisor, Dr. Tony Vannelli. Without his knowledge and guidance this thesis would not have been possible. I would also like to thank Dr. Catherine Gebotys and Dr. Miguel Anjos for their help and guidance throughout this process. Lastly, special thanks to Philip Regier, who have provided invaluable technical assistance with his insight and patience.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

This section introduces the area of routing in VLSI circuit design flow. Section 1.1.1 discusses the general VLSI design flow, and section 1.1.2 discusses the various phases in the physical design stage.

1.1.1 Digital Circuit Design Flow

As the size, computational power and functions in electronics improves, the number of digital integrated circuits contained in a chip is increasing drastically. As technology improves, the chip size, transistor density and number of transistors increases at an exponential rate. For instance, the Dual-Core Itanium 2 processor from Intel already contains over 1 billion transistors [1]. As shown in Figure 1, the number of transistors in the different generations of Intel processors grows rapidly. If the industry continues to follow the prediction of Moore's law, the number of transistors on a chip will be doubled every 1.5 to 2 years.

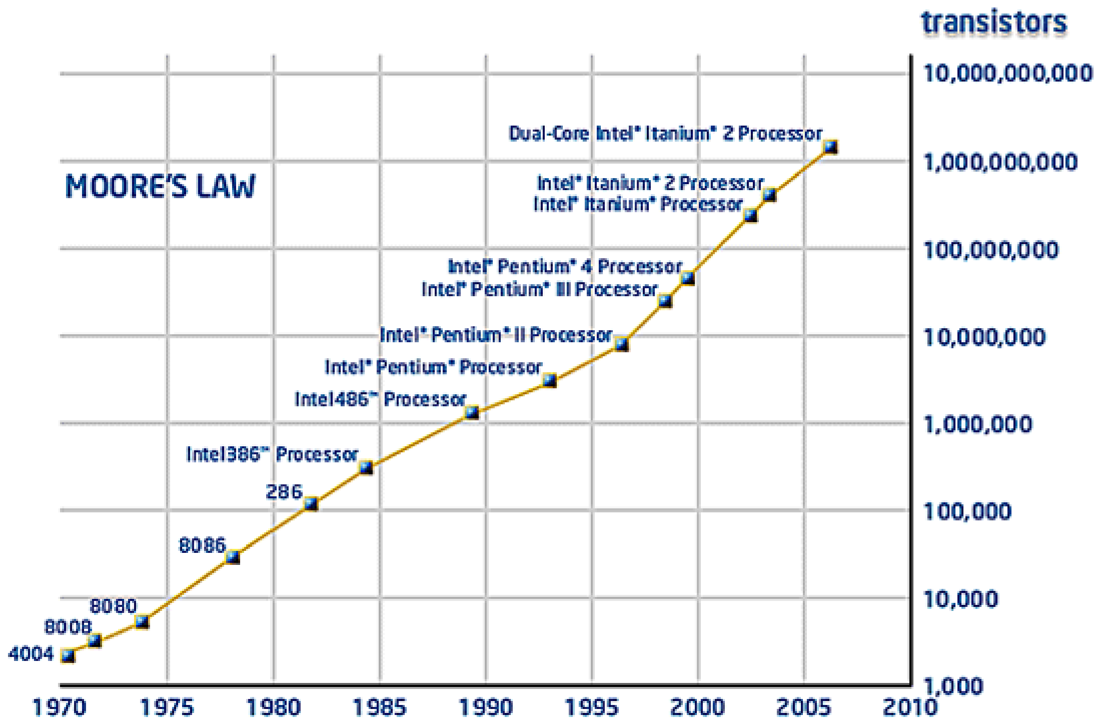


Figure 1 - The number of transistors on Intel's processors is increasing rapidly in every generation. [1]

As the circuit size increases, the design complexity of the hardware requires designers to utilize Computer Aided Design (CAD) tools to automate various parts of the design process. Because the design problem is complicated and the problem size is very large, it is necessary to break down the design process into different stages. A typical Very Large Scale Integrated Circuit (VLSI) design flow is shown in Figure 2.

In the first step, the functional requirements of the chip are specified. In addition, constraints such as the required minimum computation speed, maximum power consumed, and chip size are decided. Then, the required functions are defined with Register Transfer Level (RTL) language, as specified in the specification phase. In the third step, these functions are then translated to logic equations. The functionalities are described with basic logic operations such as NAND and XOR. These equations are then translated to actual circuits, mapping to the targeted technology available to the designer.

At physical design time, the designers perform partitioning, floorplanning and placement on the actual circuits. Then, routing is performed to interconnect different blocks of the design together. Lastly, the design would be sent to a fabrication company for actual manufacturing. Note that, several iterations of revising might be required during the design. For instance, the routing result might be fed back to the floorplanning or placement stage to obtain better results, such as lower maximum delay and area.

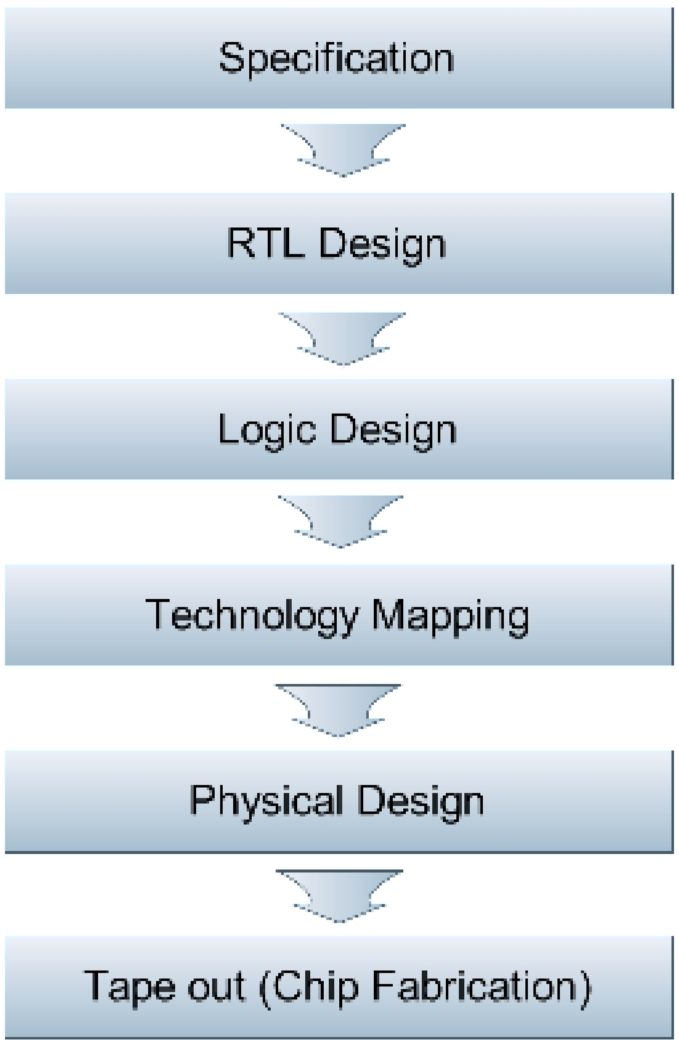


Figure 2 - Typical VLSI design flow.

As the number of transistors increases in VLSI design, so does the size of the corresponding routing problem and the time needed to solve them. One way to solve the routing problem is to model it as a linear programming (LP) problem, and utilize the interior point method to obtain the routing solution [2]. As the size of the problem increases, in order to solve the LP problem efficiently one needs to look at ways of optimizing the interior point method. Efficient methods must be used to try to solve the routing problem quickly. This is the main focus of this thesis.

1.1.2 Partitioning, Placement, and Routing

The physical design step is a complex problem of transforming a design to actual circuits, defining all the components' locations and interconnections. Because the problem is NP-hard [3], the physical design task is divided into several steps to reduce complexity and computation time. As shown in Figure 3, they are partitioning, floorplanning, placement, and routing. An important characteristic of this approach is that the result of one stage is highly dependent on the quality of the previous stage. For instance, the congestion in the routing stage is dependent on the location of the modules, which is determined in the placement stage.

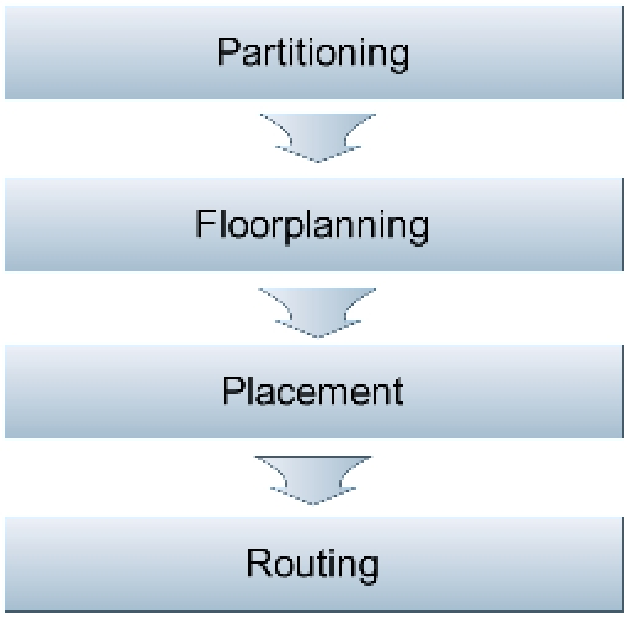


Figure 3 - The steps required in physical design.

As mentioned in section 1.1.1, the number of transistors is growing exponentially every year. This increase in the size and complexity of the problem makes the layout problem much harder to solve. In the partitioning stage the goal is to divide the circuit into different sub-blocks which are intra-related and of a smaller size, such that the problem can be solved in reasonable time. Various factors such as the size and number of sub-blocks, and the number of interconnections between the sub-blocks are considered when partitioning is performed [4]. Because the partitioning problem is NP-hard [5], constructive and iterative heuristics are used to solve the problem [6]. An example of circuit partitioning is shown in Figure 4. In this figure, the circuit is partitioned into five sub-blocks. The number of interconnection from sub-block D to other sub-blocks is three.

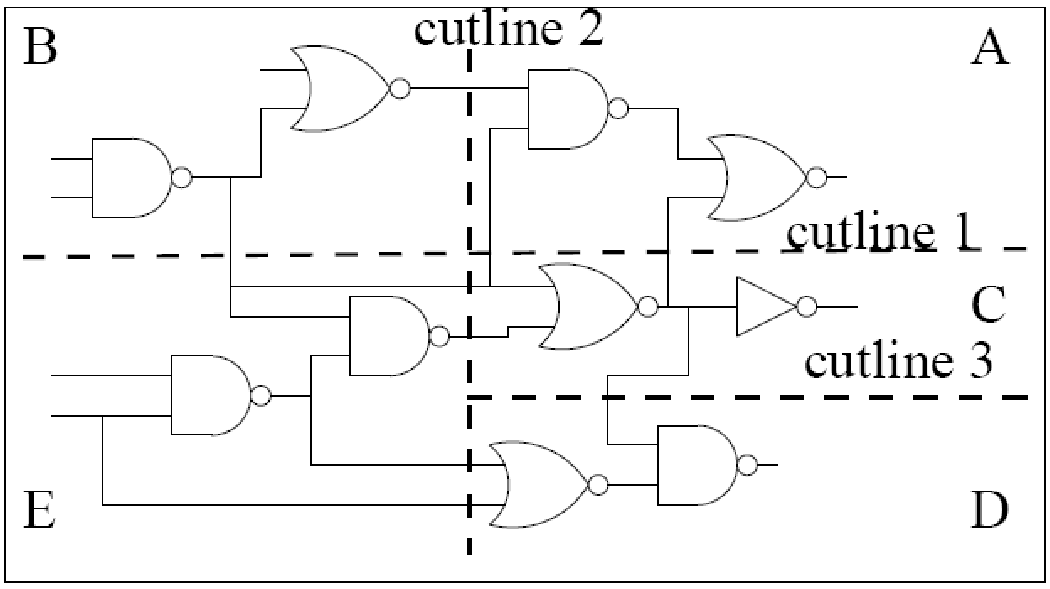


Figure 4 - An example of circuit partitioning. [4]

After the partitioning stage, the divided blocks and the interconnections required are passed down to the floorplanning phase. The floorplanning program would first determine the relative position of the sub-blocks. Sub-blocks with a higher number of interconnections between them would be placed closer together. Then, the actual length and width of each sub-block are determined. Most often the goal of the floorplanning stage is to optimize the area and wirelength, subject to various constraints such as no overlap between module, and the available chip area defined in the specification stage. Also, the resulting floorplan must be routable. Other optimization goals such as power and delay are also considered. The floorplans can be classified as variable-die (chip dimension is variable) or fixed-die (chip dimension is fixed). Because both of these problems are NP-hard, approaches such as constructive, iterative, and mathematical programming have been used to solve the floorplanning problem [6].

After the floorplanning, the actual location of each of the sub-blocks must be determined. At this placement stage, the goal is to minimize objectives such as wirelength and area, while keeping the routability in mind. For instance, minimizing area can create highly congested area, which affects the routability in the routing stage. Since the placement problem is also NP-hard, approaches such as simulated annealing, min-cut, force-directed, evolution, numerical optimization, and convex optimization methods one used in the past to solve the placement problem [5].

After the location and dimensions of the sub-blocks are determined, the interconnections need to be established. Usually, the routing stage is divided into two stages. In the global routing stage, the approximate regions for each of the interconnections to go through are determined. Then, at the detailed routing stage, the exact geometries of the interconnections are decided [4]. The goal of the routing stage is to optimize objectives such as wirelength, maximum delay, area, number of vias, power, and congestion, subject to constraints such as congestion and maximum delay. Even though the routing stage is divided into two stages to reduce complexity, the routing problem is still NP-hard [6]. Various approaches such as linear programming, sequential routing, and meta heuristics are used to solve the routing problem efficiently. In addition, to reduce the complexity of the problem, often multi-layer hierarchical approaches are used [4].

1.2 Research Motivations

This section discusses the motivation for the research, and the research approaches. Section 1.2.1 discusses the reason for the need of an efficient routing methodology, and section 1.2.2 explains the difficulties and solution in solving the routing problem using optimization methods. Section 1.2.3 gives an overview of the proposed research.

1.2.1 Global Routing

As the number of transistors increases dramatically, the number of interconnects in a chip grows accordingly. Also, since the transistor sizes decreases in deep submicron regime, but the chip area is staying the same or often growing, the interconnects are becoming an ever more important factor in determining the various performance parameters such as delay, power, and thermal consumption. This makes efficient optimized routing for interconnects an important step in the design flow.

Because the number of transistors is increasing, the complexity of the routing process is growing as well. Additionally, contemporary VLSI design often requires routing to fulfill various conflicting objectives. For instance, one might want to simultaneously optimize for delay, power, area and temperature (ie. avoiding EM effects and hotspots). This requires one to optimize the various parameters such as the number of via bends, maximum wire congestion, wirelength, and consumed power. However, since the parameters are inter-related and often conflicting, achieving these objectives simultaneously becomes difficult. Thus, the routing process becomes very time consuming, especially to the VLSI industry where time to market is critical for success. Clearly an efficient method to perform the multi-objective global routing is crucial for future VLSI designs.

1.2.2 Solving the Linear System Generated by Routers

One of the research directions in efficient routing is to translate the problem into an optimization problem. Then, one can apply an interior point method to attempt to solve the system. Even though the interior point method is able to solve the problem in polynomial time [2], the process is still very time consuming. Various preprocessing techniques and optimizations such as preprocessing and problem reduction are done, such as the work described in [2] and [7]. However, the critical bottleneck in the interior point method is that at each iteration it is required to solve yet another linear system (ie. The projection step). One of the successful research directions in solving this inner linear system is to utilize the power of parallel computation. Because the underlying mathematical model of the routing problem is a large sparse linear system, the operations required are often parallelizable. Thus, using multiple processors can allow for speedup due to parallel computations. In the case of shared memory system, the decrease in communication costs compared to networked system gives room for great speedup to be obtained. This provides a good methodology to tackle the ever-increasing size and complexity of the routing problem.

1.2.3 Overview of Proposed Research

The objectives of the proposed research are to solve the routing problem efficiently in the following way. After placement, a set of possible routing trees and design constraints used, along with an objective function that considers the relative importance of various design objectives are used to guide the routing phase. This data is transformed to an optimization model so that an interior point method can be applied. Before the interior point method, preprocessing will be done to reduce the problem size. The problem is then fed into the interior point method to solve. During the interior point computation, further optimizations will be done to reduce the problem size. As well, the inner linear system in each of the interior point loop will be solved by a parallel solver package called *PSPASES[8]*. Lastly, the solution obtained will be rounded off to zeros and ones to represent the final choices on the routing trees.

1.3 Thesis Contributions

The contributions of this thesis are as follows:

1. Integrated preprocessing, optimization, and randomized rounding techniques for the interior point approach to solve the routing problem.
2. Proposed a new optimization that further decreases the time required for interior point iterations.
3. Investigated the changes required in the model, interior point method, and randomized rounding to adapt to some new features. This includes calculating the maximum edge congestion dynamically and usage of the new optimization.
4. Investigated the effects of utilizing parallel solvers to solve the inner linear system in the interior point method efficiently.

1.4 Thesis Organization

Chapter 2 provides the background information in modeling the routing problem, the details of the interior point method to solve the problem, and past research on parallel solvers. Furthermore, techniques on preprocessing the problem and optimizations on reducing the problem size are discussed. Chapter 3 discusses the proposed new optimization, and the required changes in the model, interior point method, and randomized rounding. Chapter 4 discusses the parameter values found experimentally to apply the optimization efficiently, and the results obtained. Finally, Chapter 5 discusses conclusions and possible future work.

2 Routing Background and Methodologies

2.1 Global Routing

In general, global routing is the problem of finding the interconnection paths between parts of the sub-blocks. The input to global routing is the location of the sub-blocks and the position of the terminals (input/output port from the sub-blocks). In addition, information about which terminals need to be interconnected (netlist) is given. The goal of global routing is to find the interconnection paths such that various objectives such as estimated maximum delay, total wirelength, area, congestion, etc are optimized.

The routing problem is usually modeled as a graph problem. An example of such transformation for standard-cells is shown in Figure 5. The chip area is divided into different area (bins), and the cells are assumed to be in the center of the bins [4]. The transformation is done such that the vertices represent the input/output ports to the cells, and the edges represent possible paths for routing. In this graph representation, a set of vertices and edges represents interconnections from one cell to another.

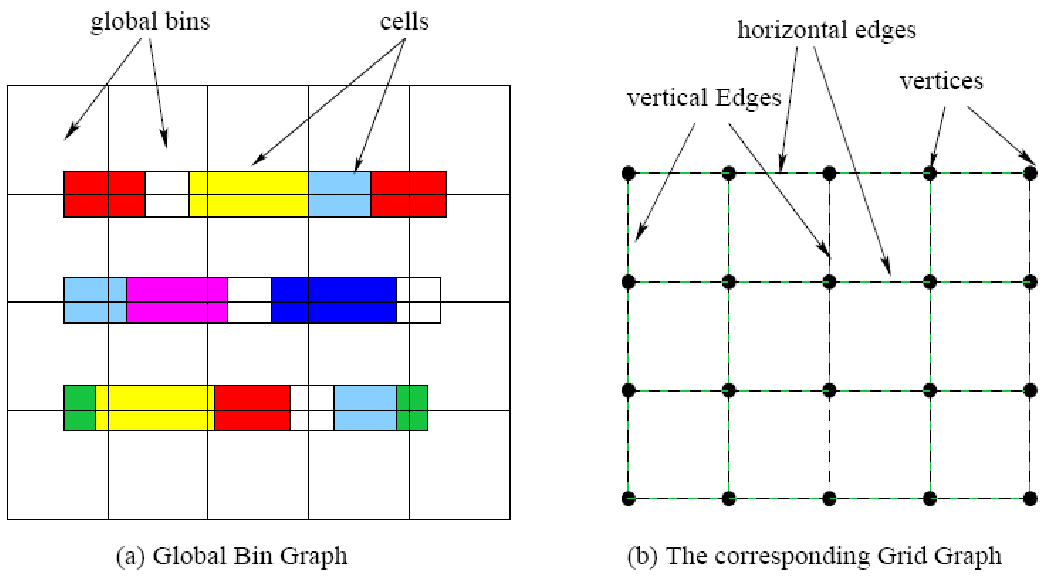


Figure 5 - Transformation from the modules to a grid graph [4].

2.2 Sequential Routing

Global routing algorithms can be classified into sequential routing and integer programming routing. In sequential routing, the nets are sequentially routed one by one. The routing sequence is determined by the relative importance of the nets. The *maze runner* algorithm proposed by [9] is a sequential routing algorithm for finding the optimal route for two terminal nets. As can be seen in Figure 6, the maze routing algorithm keeps expanding in all direction from the source terminal S, until it reaches the destination T. The disadvantages with maze routing are that it is not capable of routing multi-terminal nets. Further more, it requires a large amount of memory and computation for large graphs, because information for each vertex has to be kept and a lot of vertices are traversed [10]. Research in [11] and [12] reduces the computation time, while [13] and [14] propose line-probe algorithms to reduce the required memory. As shown in Figure 7, instead of searching one step at a time, the line-probe algorithm keep drawing lines from both the source and the destination until there are combination of lines that connects the two terminals. Both the computation time and the required memory are reduced, however the path generated is not guaranteed to be optimal. To route multi-terminal nets, various approaches such as dividing multi-terminal nets into set of two terminal nets [15] and generating Rectilinear Steiner Trees [15-21] are used.

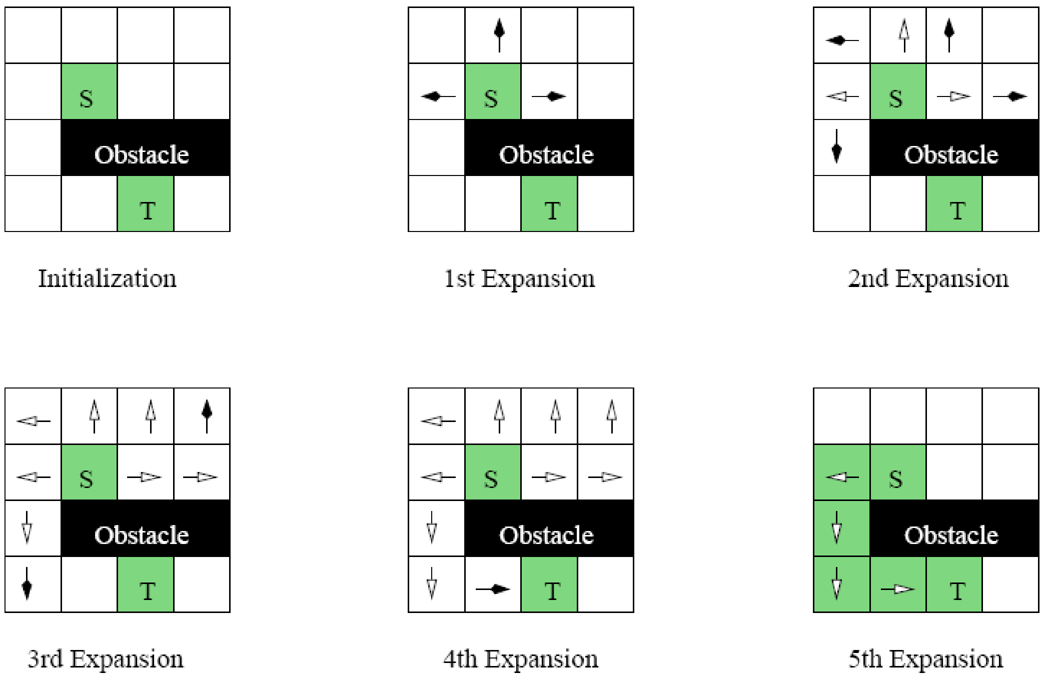


Figure 6 - Running of maze routing from terminal S to T [4].

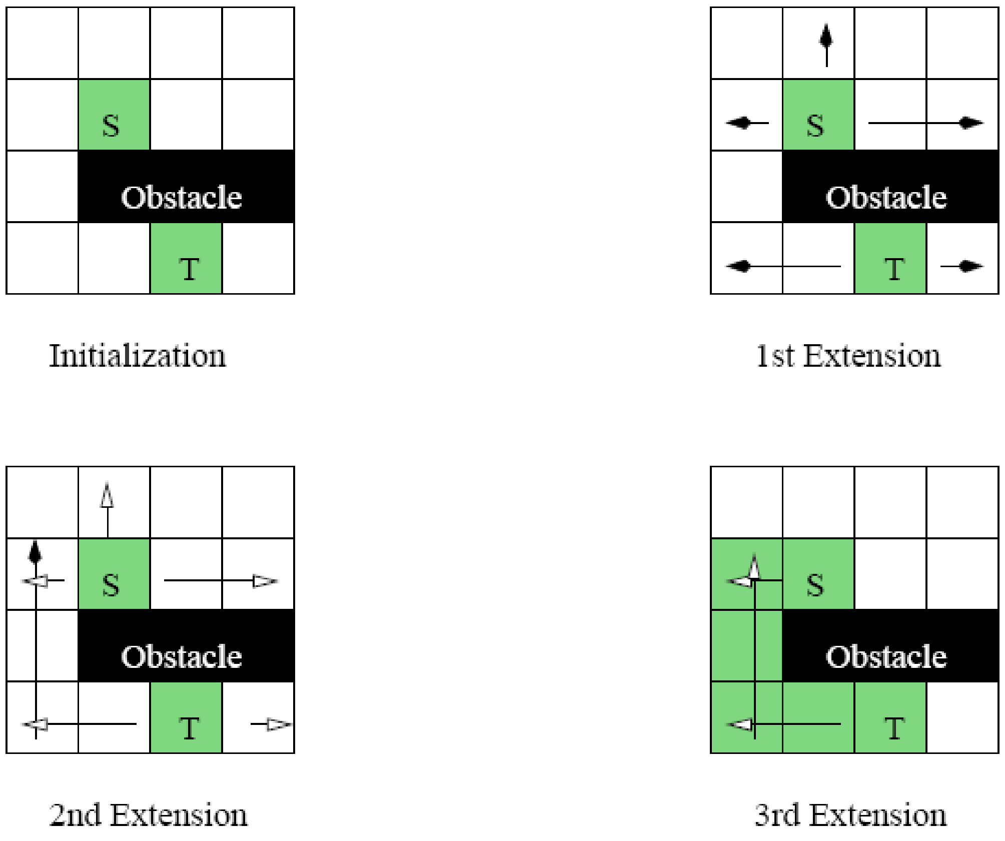


Figure 7 - Running of line probe algorithm from terminal S to T [4].

The main problem of sequential routing is that the algorithm solves the problem from a local point of view. The nature of the sequential routing makes it difficult to obtain a globally optimal solution. Because the nets are routed one at a time, the routes generated at the earlier stage can block the nets at the later stage. To solve this problem, research such as rip-up and reroute [23], which try to avoid congestion by using congestion estimation in the reroute stage [24], are proposed. Even though this research enhances sequential routing, due to the nature of the sequential method there are still some inherent disadvantages. For instance, one cannot be sure whether a feasible solution exists, and whether the obtained solution is globally optimal.

2.3 Integer Programming Routing

The second class of solvers for the routing problem is integer programming routing. In this method, the routing problem is formulated as an integer programming problem. A set of routes (trees) are generated for each net, and the problem is solved concurrently. The integer programming formulation is shown in Figure 8. In this formulation, the y_j variables are integer variables representing each of the n trees generated for the nets. A variable y_j takes on a value of 1 if this route is used, and 0 if it is not. Each of these trees are weighted by the weights b_j, which gives preference to certain trees depending on the estimated congestion, wirelength, number of vias(bends), etc. *Constraint 2.1* specifies that for each of the t nets in the set of nets N_k, at most one tree is selected to connect the net. For the second constraint, the value $a_{ij}=1$ if tree y_j passes through edge i, and equals to zero if it does not. *Constraint 2.2* states that for all the p edges, the maximum congestion caused by the trees has to be less than or equal to value c_i. In other words, *Constraint 2.1* limits each of the nets to only use one tree, and *Constraint 2.2* limits the tree choices such that the maximum congestion is not violated. The objective function is to choose as many trees as possible (ie. route as many nets as possible). The advantage of this formulation is that it resolves all the routes simultaneously, which guarantees a globally optimal solution if one exists. As shown in [4], it is also easy to incorporate multiple optimization goals into the problem, such as congestion, via bends, wirelength, and power.

$$
\begin{aligned}
& Max \sum_{j=1}^{n} b_j y_j \\
& s.t. \\
& \sum_{y_j \in N_k} y_j \le 1, k = 1.t \quad \text{(Constraint 2.1)} \\
& \sum_{j=1}^{n} a_{ij} y_j \le c_i, i = 1..p \quad \text{(Constraint 2.2)} \\
& y_j \in \{0,1\}
\end{aligned}
$$

Figure 8 - Integer programming formulation of the routing problem.

2.4 Linear Relaxation

In the integer programming formulation, the problem is easier to solve when it is relaxed to an integer linear programming problem (ILP). Then various techniques such as simulated annealing [25], column generation [26] and interior point method [2] can be used to solve the relaxed problem. Multi-objectives global routing has been investigated in [6] for simultaneous optimization of vias, congestion, and wirelength. Yang [4] researched additional simultaneous optimization for power. After the relaxed problem is solved, techniques such as choosing the tree with the highest y_j value [27] and randomized rounding [27-29] are used to obtain integer solutions.

2.5 Hierarchical and Multi-level Routing

Unfortunately, the solution time of the integer programming problem is related exponentially to the number of trees [6]. Since the problem can become large it can be very time consuming to solve the routing problem in this formulation. One approach is to divide the circuits into different parts, and solve each part separately using the integer programming formulation. This top-down hierarchical approach is able to reduce the size of the problem, but the problem might become infeasible to solve, and the computation time can be large due to the creation of many smaller integer programming problem [31]. Another approach is to use a bottom-up hierarchical approach proposed by [32]. In this case, the routing region of each sub-problem gradually gets larger as the program solves and subsequently merges the different regions.

In [33], a multi-level routing approach is proposed. As shown in Figure 9, the circuit is coarsened gradually, estimating information on routing resources in the process. A multicommodity flow algorithm is used to obtain a solution at the coarsest level, and then the problem is uncoarsened gradually. At each level of uncoarsening the solution is further improved, and finally at level 1 the solution is fed into a detailed router to obtain the final solution. The advantage of the multi-level routing approach is that it can handle large routing problems, because it performs coarsening on the problem. Furthermore, because it can obtain information on routing resources during the coarsening process, the completion rate of finding a solution is higher than hierarchical approach. Lastly, it is able to perform routing in less time.

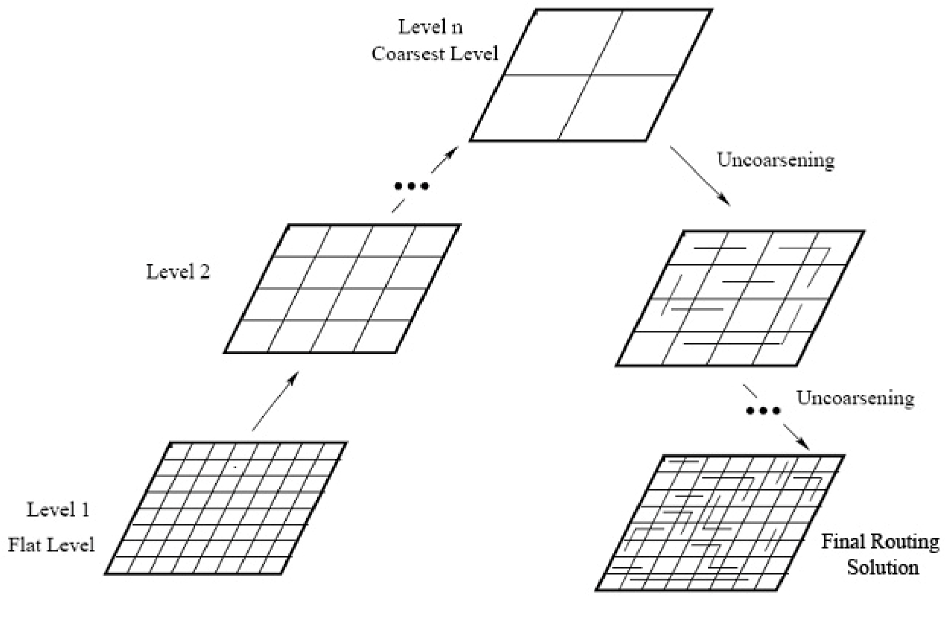


Figure 9 - Multi-level routing [4].

2.6 Tree Generation

For the integer programming formulation, in order to find the routes to use for the interconnections one needs to generate the possible routes to choose from. Since the routes in actual circuits run only horizontally and vertically, any generated route will also only run rectilinearly. Because the number of possible trees is large, initially only the minimum trees are considered [6]. This problem is usually formulated as finding the different Rectilinear Steiner Minimum Trees (RSMT) to connect the terminals. From [34], it was shown that using the Hanan grid (a grid formulated by drawing vertical and horizontal lines from the vertices of the graph), every Steiner Minimum tree can be formed. An example of the Hanan grid is shown in Figure 10.

Since the RSMT problem is NP-hard [35], one usually finds the Minimum Spanning Tree (MST) first, and then transforms the MST to a RSMT. This is because a MST can be found efficiently in polynomial time [35, 36]. As shown in [38], the ratio of the lengths between a MST and a RSMT is less than or equal to 1.5. Thus, one can ensure near optimal RSMT are generated by first generating MST, and then transforming the non-rectilinear edges to either L shape or Z shape tree [6]. This transformation is shown in Figure 10. In the figure, the original MST is rectilinearized to a rectilinear steiner tree. To ensure the number of vias is minimized, only minimum-bend trees are generated [6]. An example of minimum-bend tree and non-minimum bend tree is shown in Figure 11.

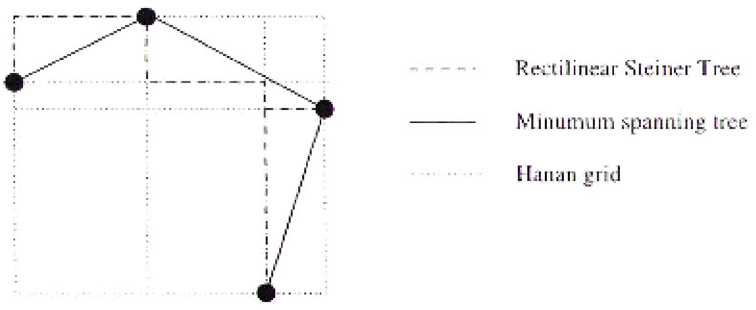


Figure 10 - Connecting the vertices using Hanan grid, RST, and MST [6].

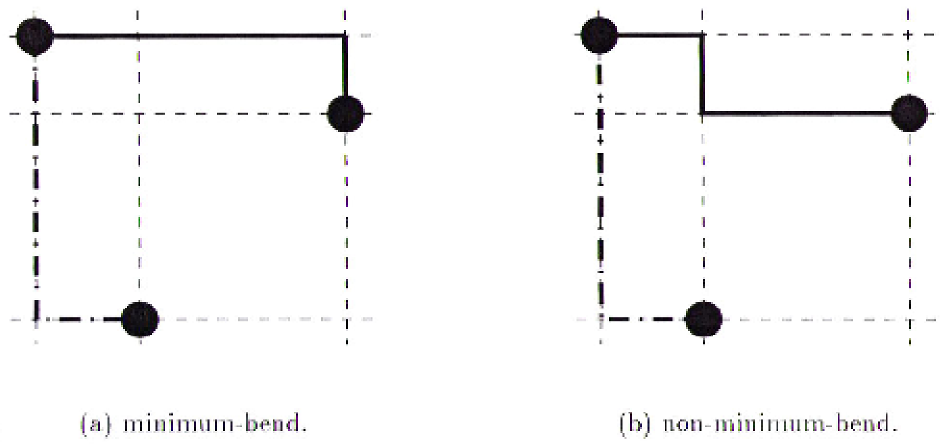
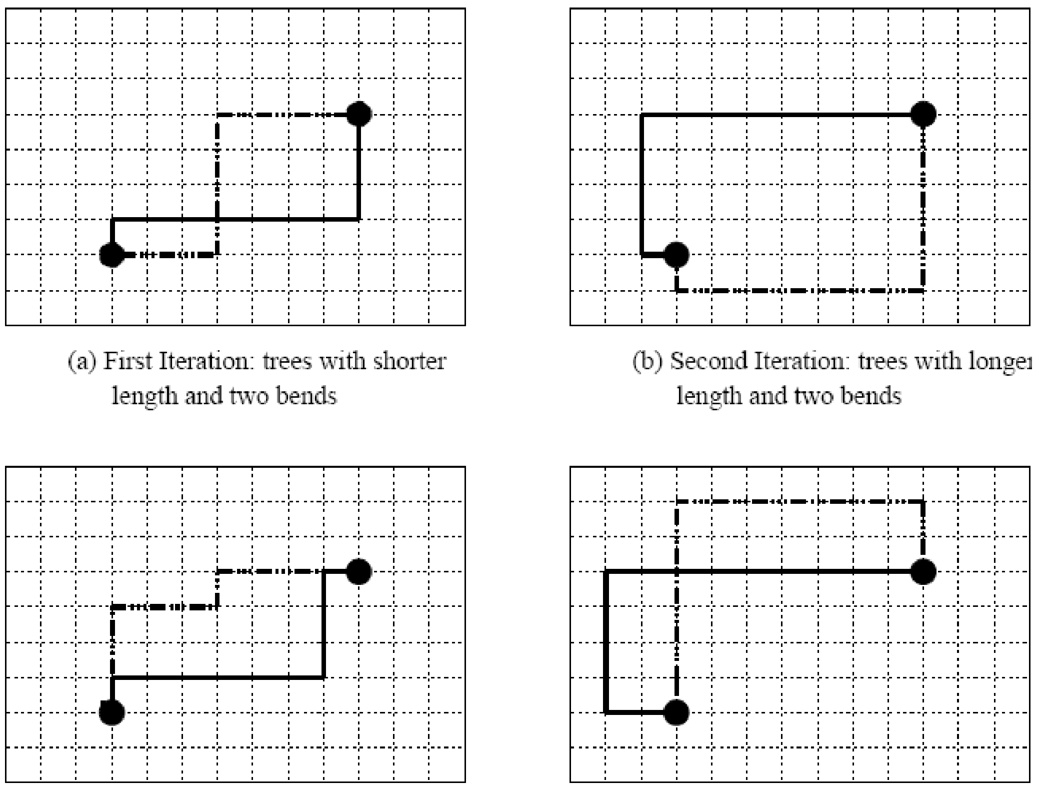


Figure 11 - Connection between 3 vertices, using (a) minimum bend trees and (b) non-minimal bend trees [6].

In the research of [6], since the number of possible minimum-bend trees grows exponentially with the number of terminals, to limit the number of tree choices the nets are divided into two categories. The nets with two or three terminals are categorized as short nets, and the nets with more terminals are categorized as long nets. Minimum-bend trees are produced for two terminals nets. On the other hand, three terminal nets are split into two sets of two terminals nets, and minimum-bend trees are generated accordingly. In the research of [4], RSMT are generated for long nets using a program called GeoSteiner.

With these generated trees, it is possible that there are no solutions that satisfy all the constraints. For instance, there are limits on the maximum congestion allowed on each of the routing paths (channels). Thus, with only the minimum trees there might be no feasible solution. To resolve this problem, [6] proposed a congestion estimation algorithm to predict upper and lower estimates of the congestion. For nets with trees that pass through congested areas, additional trees are generated to avoid congestion. Trees are iteratively added until congestion is eliminated or the algorithm reached the iteration limit. An example of generating additional trees is shown in Figure 12.


(a) First Iteration: trees with shorter length and two bends

(b) Second Iteration: trees with longei length and two bends

(c) Third Iteration: trees with shorter length and three bends

(d) Fourth Iteration: trees with longer length and two bends

Figure 12 - Generating additional trees [4].

2.7 Optimization Metrics

During the routing process, there are various conflicting optimization objectives. Firstly, it is preferable to reduce the length of the interconnect. As chip size stays constant/growing and device size scales down, interconnects are becoming increasingly dominant in terms of delay, area, power, thermal, and reliability in the deep submicron regime. Second, one wants to reduce congestion in the routing. Because of finite routing resources, some areas in the chip can be congested with trees and become fully occupied. Consequently, interconnects might have to detour from the congested region. Furthermore, congestion can cause hotspots in the chip, and even cause a design to be unrouteable. Third, it is desirable to minimize the number of vias (ie. bends in the route). This is because vias increase manufacturing cost, decrease fabrication yield, and generate higher circuit delay. Finally, in deep submicron regime, power consumption of interconnects can no longer be ignored. Thus, an important objective is to minimize the power consumed by the IR drop. Note that one optimization objective might conflict with another. For instance, to avoid congestion one might need to use the non-optimal length route. This would conflict with the objective of reducing wirelength.

2.8 Summary

This chapter formulated the global routing problem and various routing methodologies, including sequential routing, integer programming routing and linear relaxation, hierarchical and multi-level routing, and generating routing trees. We also explained the various conflicting optimization metrics involved in solving routing problems. The need for a more efficient routing method that allows for routability detection and globally optimal solution was shown. The next chapter will discuss how to address these problems, by means of formulating the problem as an optimization problem and applying various optimizations.

3 Global Routing Problem Formulation

In general, the routing problem consists of determining how to formulate paths from a source node to a destination node, subject to various constraints such as via bends, congestion, and wirelength [3]. Usually, multiple possible routes, called trees, are generated for each net. The problem is to determine which tree to select for the nets, subject to various constraints and objectives.

Essentially, the routing problem is a combinatorial problem, where one needs to make a decision of choosing a tree or not (1 or 0) for a particular net. As shown in section 2.3, this problem can be formulated as an integer programming (IP) problem. This allows the designers to obtain a global optimal solution. This chapter will describe the process of the actual linear relaxation of the IP problem. Furthermore, methods to solve the relaxed problem more efficiently, such as using the interior point method, parallel linear solvers, preprocessing, and optimization algorithms are described to further speed up the computation time. Lastly, randomized rounding is described to obtain integer solution from the fractional solution.

3.1 Modeling the Routing Problem

A sample of a routing problem can be seen in Figure 13. Each pair of source and destination nodes forms a net. The routing problem consists of determining which of the trees to use for each net in the circuit. This problem can be formulated as an integer programming problem.

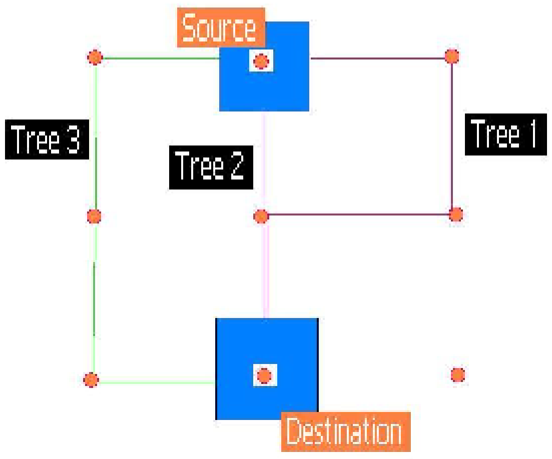


Figure 13 - The routing problem consist of deciding which tree to select to connect the source and destination node.

The integer programming formulation can be linearly relaxed to linear programming (LP) problem for more efficient solving. As suggested in [1], a routing problem with n trees, t nets, and p edges can be formulated as an LP problem as follows:

Model 1

$$Minimize \sum_{j=1}^{n} (w_{ij} + w_{bj}) y_j + \beta_z Z_{\max}$$

$$s.t.$$

$$\sum_{y_j \in N_k} y_j \le 1, k = 1,2,..t \qquad (Constraint\ 1.1)$$

$$\sum_{j=1}^{n} a_{ij} y_j - Z_{\max} \le 0, i = 1,2,..p \qquad (Constraint\ 1.2)$$

$$y_j \ge 0 \quad and \quad Z_{\max} \ge 0 \qquad (Constraint\ 1.3)$$

$$j \in \{1.n\}, 0 \le Z_{\max} \le AllowedCongestion$$

The variable y_j is the decision variable representing whether the tree j is selected or not. The weights b_j determines how preferable the corresponding tree is. *Constraint 1.1* restricts that only 1 tree can be selected for a particular net, and *Constraint 1.2* restricts that the congestion for a particular edge on the routing would not go over value c_i. Originally, y_j should be binary variables, with y_j=1 representing tree j being selected, and y_j=0 representing tree j not being selected. Since this is a relaxed version of the IP problem, it is possible to have non-integer values for y_j. We want to confine the values of y_j to be between and including one and zero. Forcing y_j to be less than or equal to 1 is done implicitly with the *Constraint 1.1*, where forcing y_j to be greater or equal to zero is done explicitly by the C*onstraint 1.3*. When y_j has a non integer value between one and zero, it defines how much it is preferred in the tree selection. For instance, $y_j = 0.6$ and $y_k = 0.1$ implies that one would likely select tree *j* over tree *k*.

Model 1 hardcodes the congestion constraints into the right hand side of the inequality. What one might want to do is to dynamically determine the congestion limit needed, based on a trade off between congestion and other objectives such as wirelength and number of via bends. This is the model proposed by [2]. With N_k being the set of nets (ie. k=1 represents net 1), and there are t nets, n trees, and p edges, the model is:

Model 2

$$Minimize \sum_{j=1}^{n} (w_{lj} + w_{bj}) y_j + \beta_z Z_{\max}$$

$$s.t.$$

$$\sum_{y_j \in N_k} y_j \le 1, k = 1,2,.t \quad \textit{(Constraint 2.1)}$$

$$\sum_{j=1}^{n} a_{ij} y_j - Z_{\max} \le 0, i = 1,2,..p \quad \textit{(Constraint 2.2)}$$

$$y_j \ge 0 \quad and \quad Z_{\max} \ge 0 \quad \textit{(Constraint 2.3)}$$

$$j \in \{1.n\}, 0 \le Z_{\max} \le AllowedCongestion$$

In this model, w_{lj}, w_{bj}, and β_z are the weights for the wirelength, number of via bends, and congestion, whereas Z_{max} represents the maximum congestion in the system. a_{ij} represents whether tree j occupies edge *i*. This model has several advantages. Firstly, the various conflicting objectives can be modeled in the objective function. Second, the determination of Z_{max}'s value is guided by the corresponding weight of Z_{max} in the objective function. The maximum congestion needed can be dynamically calculated in the problem instead of hard coded.

This model can be translated to matrix form as shown in Figure 14. The matrix **A** represents the left hand side of the constraints from *Model 2*. Each of the first *n* columns represents y_j (the trees), and the last column represent the variable Z_{max}. The first *t* rows of the matrix represents *Constraint 2.1*, which states that for each of the *t* nets only one tree can be chosen. The subsequent *p* rows represents *Constraint 2.2*, which states that the congestion of each of the *p* edges is less than or equal to Z_{max}. The last *n+1* rows represents *Constraint 2.3*, which states that all the y_j and Z_{max} variables have to be greater or equal to zero.

$$Minimize \sum_{j-1}^{n} (w_{ij} + w_{bj}) y_j + \beta_z Z_{\max}$$

such that

$$\mathbf{Ay} \le \mathbf{b}$$

where

$$A = \left[\begin{array}{ccccc|c} \multicolumn{5}{c|}{Trees} & Z_{\max} \\ 1 & 1 & & & & \\ & & 1 & 1 & 1 & \\ - & - & - & - & - & - \\ 1 & 1 & 1 & & & -1 \\ & & 1 & 1 & 1 & -1 \\ 1 & & 1 & & & -1 \\ & 1 & & 1 & & -1 \\ - & - & - & - & - & - \\ -1 & & & & & \\ & -1 & & & & \\ & & -1 & & & \\ & & & -1 & & \\ & & & & -1 & \\ & & & & & -1 \end{array}\right] \begin{array}{l} \\ \} \; t \text{ rows of Constraint 2.1} \\ \\ \\ \\ \} \; p \text{ rows of Constraint 2.2} \\ \\ \\ \\ \\ \\ \} \; n{+}1 \text{ rows of Constraint 2.3} \\ \\ \\ \\ \end{array} \quad B = \left[\begin{array}{c} 1 \\ 1 \\ - \\ 0 \\ 0 \\ 0 \\ 0 \\ - \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array}\right]$$

Figure 14 - Model 2 translated in Matrix form.

3.2 Karmarkar's Interior Point Method

As suggested in [2], we can relax the IP routing problem to a LP problem, and solve it using interior point methods. The advantage in solving the problem this way is that the number of iterations required is independent of the problem size, and it is a polynomial time algorithm [2]. In contrast, the standard Simplex algorithm for solving LP problems can require an exponential number of iterations, although typically it is polynomial in running time. In practice, the interior point method is polynomial in terms of number of trees to choose from, and so the problem formulation is still large in size.

Interior point methods are used to solve linear programming problems. One of the most important breakthroughs in this area is the polynomial-time interior point method in [39]. Vanderbei *et al.* [40] and Barnes [41] developed primal-affine algorithms, which use linear transformation instead of projective transformation. Adler *et al.* [42] developed a dual-affine scaling algorithm, which solve the dual version of the problem instead. Kojima *et al.* [43] developed a primal-dual interior point algorithm, which solves the linear programming problem more efficiently. Later on, improvements to the primal-dual algorithm such as [44] were developed. The primal-dual algorithm merges the constraints of the problem into the objective function by using a logarithmic barrier function. Then, the logarithmic barrier term is decreased at each iteration. A Lagrange-Newton method is then used to solve the problem [45]. To improve the efficiency of this method, Predictor-Corrector method in [44] uses higher order terms in the Lagrange-Newton method rather than just the first-order term to increase accuracy.

As can be seen from the ILP formulation in section 3.1, there can be many more constraints than trees (ie. many edges). As shown in the left side of Figure 15, the Simplex algorithm can be slow in solving such problems, because the iteration jumps from one constraint corner point to another [2]. If the number of constraints is large, the solution time will be inefficient. In contrast, as can be seen from the right side of Figure 15, the interior point method iteratively traverse from one point in the interior of the constraint area to another. Initially, a feasible solution is determined that is within the constrained area. Spheres are created inside the constrained area, and a nonlinear transformation is used to project to the next interior point. This iterative process continues until the difference between the product **Ay** and the vector **b** (ie. the error) is small enough [39]. Furthermore, research in [46] shows that the interior point method can be adapted to construct ellipsoids instead of spheres to accelerate the interior point projection.

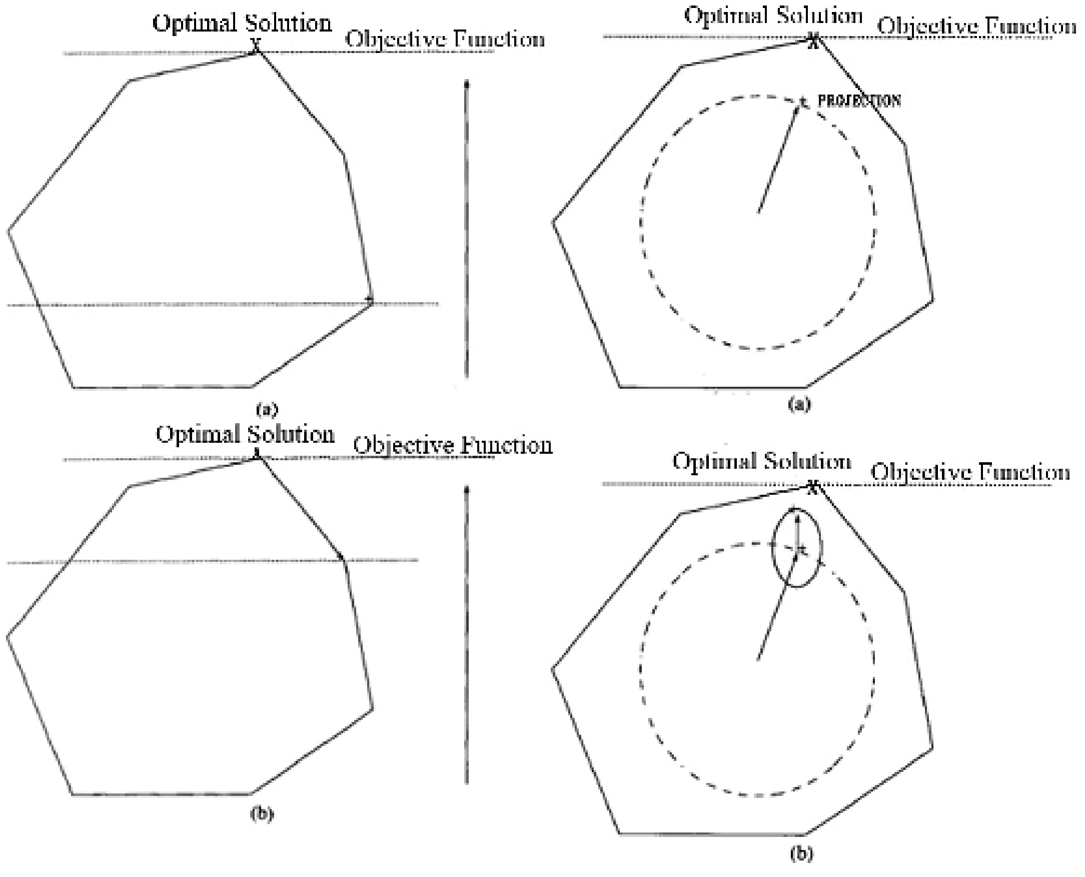


Figure 15 - Two iterations of the Simplex method (left), and interior point method (right) [2].

This thesis uses Karmarkar's dual-affine version of the interior point algorithm [42] to solve linear systems such as the *Model* 2 introduced in section 3.1. Let ***c*** be the coefficients of the objective function such that we are trying to minimize the vector product ***c x***, *exitCriterion* be some number we set to indicate the criterion for exiting the iterations, $\mathbf{x}^0$ and γ be given such that:

$$\boldsymbol{Ax^0} < \boldsymbol{b},\ 0< \gamma<1$$

The algorithm runs as follow:

```
k=-1
do {
   k = k+1
   $\mathbf{v}^k = \mathbf{b} - \mathbf{A}\mathbf{x}^k$
   $\mathbf{D}_k = \text{diag}\ \{\mathbf{v}_1^{-k}, \mathbf{v}_2^{-k}, .., \mathbf{v}_m^{-k}\}$
   $\mathbf{dx} = (\mathbf{A}^T\mathbf{D}_k^2\mathbf{A})^{-1}\ \mathbf{c}$
   $\mathbf{dv} = -\mathbf{A}\ \mathbf{dx}$
   $\alpha = \gamma \min\{ -v_i^k/dv_i \mid dv_i<0, i=1..m\}$
```

```
        $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha\,\mathbf{dx}$
        $exitCondition = |\mathbf{c}^T\mathbf{x}^{k+1} - \mathbf{c}^T\mathbf{x}^k| / |\mathbf{c}^T\mathbf{x}^k|$
    } until ($exitCondition < exitCriterion$)
```

The initial point $\mathbf{x}^0$ is defined as:

$x_j^0 < c_{min} / row_{max}$, j=1..n

$c_{min} = \min\{ c_i \}$

row_{max} = the maximum number of trees that can occupy any edges in any nets.

As can be seen from the algorithm, at each iteration the linear system:

$$\mathbf{dx} = (\mathbf{A}^T\mathbf{D}_k^2\mathbf{A})^{-1}\,\mathbf{c}$$

needs to be solved. This step is required to find the projection to the ellipsoid's boundary. As solving this linear system requires a large amount of computation, this step becomes the bottleneck in the interior point iteration. Since the algorithm might iterate 20 to 40 times [2], to solve this linear system efficiently it may be useful to utilize a parallel solver.

3.3 Previous Work on Parallel Solvers

When one analyzes the linear system that comes from interior point solving a routing problem: $(\mathbf{A}^T\mathbf{D}_k^2\mathbf{A})\ \mathbf{dx} = \mathbf{c}$, it can be noted that $\mathbf{D}_k^2$ only has values on the diagonal. Thus, $\mathbf{A}^T\mathbf{D}_k^2\mathbf{A}$ is symmetric positive definite if $\mathbf{A}$ is full row rank [2]. This is an important property to consider when deciding which technique to use to solve this linear system. In the past, there were attempts to use alternatives other than Cholesky factorization and Gaussian elimination to solve this type of linear system. For instance, in the research in [2], a Conjugate Gradient (CG) approach using an incomplete factorization method to obtain the preconditioner was used. In the past, there were various attempts at parallelizing the CG algorithm. Unfortunately, most of the research has been done on how to parallelize for specific structures of the **A** matrix which are not applicable to our problem. For instance, [47] looks at sparse block-tridiagonal matrix, and concluded that under a message passing parallel architectures, ordering of the matrix needs to be done for a parallel version of CG to be scalable. Decker *et al.* [48] investigated a parallel CG approach for dynamic simulations in power systems, which is also in block-tridiagonal matrix form. In [49], attempts to reduce synchronization overhead in the CG algorithm were also proposed. Jordan and Bycul [50] investigated the effect of parallelizing the CG algorithm by using a row-based distribution for the dense matrix and vector. Later on, [51] investigated the effects of the same approach for sparse matrix, using a storing mechanism such that only non-zero entries are recorded. It was concluded that for a distributed memory system, the speedup is less than one because the communication cost was too high for sparse matrices.

Bycul *et al.*'s [51] empirical results were based on really sparse matrices, with one test problem being a six-diagonal matrix. One of the main reasons for the unsuccessful results were due to the extremely high sparsity of the matrix, causing the communication cost to far exceed the calculations required. However, for the problem we are trying to tackle, the sparsity is not as severe as the tested problem. One sample of the sparsity pattern in the $\mathbf{A^T D_k^2 A}$ matrix can be seen in Figure 16, for a routing problem size of 262 variables (trees). As can be seen from the sparsity pattern, the matrix is sparse but it still contains a fair amount of non-zeros (14.57% non-zero entries). This means there is a higher chance for us to utilize the benefits of parallel computing with incurring much communication cost. Moreover, the machine used for this work is a shared memory multi-processor system, which will further mitigate the effects of the communication cost.

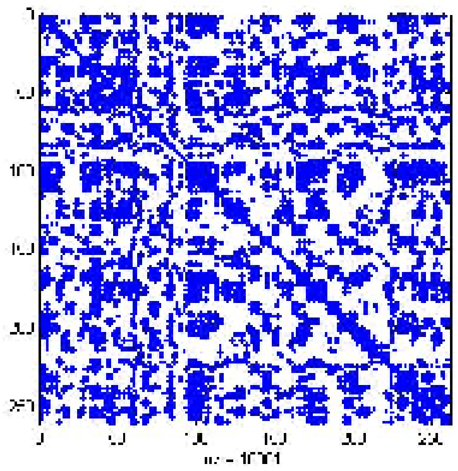


Figure 16 - Sparsity pattern of the $A^T D_k^2 A$ matrix, with problem size of 262 rows.

A successful parallel solver for sparse linear systems with symmetric positive definite matrix is the *PSPASES* (*Parallel SPArse Symmetric dirEct Solver*) package[8]. The fill reduced ordering uses a parallel multilevel nested dissection algorithm [52], which gives orderings appropriate for parallel factorization. The symbolic and numerical factorization phase uses a parallel multifrontal Cholesky factorization [53], which has been shown to be scalable and efficient. The *PSPASES* implementation details can be seen in [54]. It allows any library which follows the MPI (Message Passing Interface) standard to be used for communication, and any BLAS (Basic Linear Algebra Subprograms) library to be used for vector and matrix operations. The *parMETIS* package is used to perform parallel graph partitioning and fill-reduced ordering. Since the PSPASES package has not been used to solve the routing problem using the linear programming formulation in the past, it is worthy to examine the effectiveness of utilizing this package on solving the interior point loop.

3.4 Optimization Efficiencies

This section explains techniques that allow speeding up the solving time of the interior point method. Section 3.4.1 discusses the Remove Constraint Optimization, and section 3.4.2 explains a preprocessing technique to reduce the size of the problem before starting to solve.

3.4.1 Remove Constraint Optimization (RCO)

To speed up the iteration time of interior point method, a method was proposed in [2] to reduce the problem size as we iterate through the algorithm. The idea is that as we progress through the iterations, the y_j values are assumed to be getting closer to the solution. In such case, one can determine which congestion constraints will be inactive and remove them. For instance, with $y_1 = 0.2$, $y_2 = 0.8$, $y_3 = 0.6$, Z_{max}=1.7:

$$y_1 + y_2 < Z_{max} \Rightarrow (0.2+0.8) < 1.7$$
$$y_1 + y_2 + y_3 < Z_{max} \Rightarrow (0.2+0.8+0.6) < 1.7$$

We can see that the first constraint is not *active*, as the sum of y_1 and y_2 is not close to Z_{max}. Assuming that we have progressed enough through the iterations, the y_j values may have stabilized to a point such that there would not be major changes in the values. In such a case, by removing such non-active constraints, one can reduce the size of the problem and speed up the iterations [2]. Since *exitCondition* indicates the amount of progress from one iteration to another, when *exitCondition* is small, it indicates that one is getting closer to the solution. Thus, *exitCondition* is a good indicator for when one can start to remove non-active constraints. This threshold needs to be determined experimentally. If *exitCondition* is too large (ie. the interior point loop is still making progress in large steps), constraints might be incorrectly removed because the solution has not settled yet. If *exitCondition* is too small, then there will be time wasted when the interior point method is calculating with unnecessary constraints.

It is also necessary to determine the condition for when to remove a certain constraint. In [2], since *Model 1* in section 3.1 is used, a hard coded value was used for determining this condition. An interpretation of its method is that when any row j of *Constraint 2.2* has $a_{ij}y_j$ less than a certain value (say 50), then row j can be removed. Ie.

$$\textit{(Constraint 2.2)} \quad \sum_{j-1}^{t} a_{ij} y_j \leq c_i$$

$$\downarrow$$

$$\textit{RemoveCondition:} \quad \sum_{j-1}^{t} a_{ij} y_j < 50$$

$$\textit{where} \quad 50 < c_i$$

However, since the model used in this research is the dynamic constraint model (*Model 2*), this method would not be applicable. This is because Z_{max} is a constantly changing value, therefore one cannot hardcode a value without knowledge of what Z_{max} would be like relative to the congestion until the solver is finished. A better remove condition would be if the *slack* (difference between Z_{max} and the congestion) is greater than a certain percentage of Z_{max}, then the constraint can be removed.

$$\textit{(Constraint 2.2)} \quad \sum_{j=1}^{t} a_{ij} y_j - Z_{\max} << 0$$

$$\downarrow$$

$$\text{Remove Condition:} \quad \sum_{j=1}^{t} a_{ij} y_j << Z_{\max}$$

$$\downarrow$$

$$\text{Remove Condition:} \quad \left| Z_{\max} - \sum_{j=1}^{t} a_{ij} y_j \right| < \textit{ConstraintTolerance}$$

$$\textit{where ConstraintTolerance} = Z_{\max} * \textit{ConstraintTolerancePercentage}$$

So, in this example, if *ConstraintTolerancePercentage*=20%, Z_{max}=50, then a constraint row *j* is removed when the difference between Z_{max} and congestion is more than 10 (20% of Z_{max}). This *ConstraintTolerancePercentage* value needs to be determined experimentally. If the *ConstraintTolerancePercentage* is set too high, the constraints that are actually active and necessary will be incorrectly removed. If the value is set too low, there would be constraints left that could be removed.

3.4.2 Preprocessing

Before the routing problem is fed into the interior point method, various preprocessing techniques can be applied to reduce the size of the matrix **A**. The goal is to reduce the matrix size as much as possible to shorten the interior point method's computation time. By default, most solvers such as CPLEX would apply many preprocessing techniques. The goal in this section is to examine techniques that require knowledge of the routing problem itself.

Hare *et al.* [7] discussed a property that can be utilized to reduce matrix **A**. Firstly, it utilizes a classical preprocessing technique of removing singleton trees, which already reduces the size of matrix **A**. If there is only one tree generated for a particular net (singleton tree), then this tree and the according constraints can be removed, because the net has to pick this tree for routing. Then, it utilizes a property derived from removing the singleton tree to further reduce the matrix size. Using *Model 2* from section 3.1, the operations are as follow:

1. Find rows in *Constraint 2.1* ($\sum_{y_{j\in N_k}} y_j = 1$), such that there is only one y_j=1 (singleton trees). Say these *m* singleton trees exist, ie. the set $T_k = \{\forall j \quad j = singleton\ tree\}$. We set y_j=1 for all j in T_k. (We are picking all the singleton trees as our choices.)
2. Remove *Constraint 2.1* and *Constraint 2.3* for these trees.
3. Since we are going to use these singleton trees, the singleton trees are going to create congestion. Before we remove these singleton trees from the matrix **A**, we need to put the congestion that these trees occupy into the constraint. Therefore, in *Constraint 2.2* ($\sum_{j=1}^{n} a_{ij} y_j - Z_{\max} \le 0, i = 1..p$), for all *i*, subtract a_{ik} from the right hand side, where *k* is in set T_k. Ie.

$$\textit{(Constraint 2.2)} \sum_{j=1}^{n} a_{ij} y_j - Z_{\max} \le 0, \quad i = 1..p$$

$$\downarrow$$

$$\sum_{\substack{j=1 \\ j\notin T_k}}^{n} a_{ij} y_j - Z_{\max} \le 0 - \sum_{k\in T_k} a_{ik}, \quad i = 1..p$$

$$\downarrow$$

$$\sum_{\substack{j=1 \\ j\notin T_k}}^{n} a_{ij} y_j - Z_{\max} \le -\sum_{k\in T_k} a_{ik}, \quad i = 1..p$$

4. We can now remove the singleton trees from the matrix. Since the appropriate rows are removed from Step 2, we now only need to remove the columns that represents these singleton trees. We remove all columns *k* where *k* is in set T_k

from the matrix **A**.

5. From the resulting relationship in Step 3, [7] noticed the following:

$$\sum_{\substack{j=1\\ j\notin T_k}}^{n} a_{ij}y_j - Z_{\max} \le -\sum_{k\in T_k} a_{ik},\quad i=1..p$$

$$\downarrow$$

$$Z_{\max} - \sum_{\substack{j=1\\ j\notin T_k}}^{n} a_{ij}y_j \ge \sum_{k\in T_k} a_{ik},\quad i=1..p$$

$$\downarrow$$

$$Z_{\max} \ge \sum_{k\in T_k} a_{ik},\quad i=1..p$$

$$Since\quad y_j \ge 0, a_{ij} \ge 0$$

$$\downarrow$$

$$\therefore Z_{\max} \ge \max_i \left\{ \sum_{k\in T_k} a_{ik} \right\},\quad i=1..p$$

6. From Step 5, a lower bound of Z_{max} is obtained. It states that the maximum congestion Z_{max} has to be greater than the maximum congestion on any edge i that is occupied by the set of singleton trees T_k. This allows us to remove any edge constraint (*Constraint 2.2*) for which the maximum congestion is less than the maximum edge congestion that is created by the singleton trees. Suppose S is all set of $\{y_j\}$ such that $\{y_j\}$ is a valid combination of tree choices, ie. only one tree is chosen for each net. $E_{max\, i}$, the maximum congestion of a particular row i of *Constraint 2.2*, is calculated as follow:

$$E_{\max_i} = \max_{y_j\in S}\left\{ \sum_{j=1}^{n} a_{ij}y_j \right\}$$

7. Let C_{max} be the maximum congestion caused by the singleton trees on any edges. From Step 5 and 6, the constraint elimination criteria is summarized as follow:

a) $$Let\quad C_{\max} = \max_i \left\{ \sum_{k\in T_k} a_{ik} \right\}\quad i=1..p$$
$$\therefore Z_{\max} \ge C_{\max}$$

b) *(Constraint 2.2)* $\sum_{j=1}^{n} a_{ij}y_j - Z_{\max} \le 0$

$$\downarrow$$

$$\sum_{j=1}^{n} a_{ij} y_j \leq Z_{\max}$$

$$\downarrow$$

$$\sum_{\substack{j=1 \\ j \notin T_k}}^{n} a_{ij} y_j \leq E_{\max_i} \leq Z_{\max}$$

c) $\therefore \quad \mathit{if}\ E_{\max_i} < C_{\max} \leq Z_{\max}$

Remove Row i.

3.5 Randomized Rounding

At the end of the interior point loop, the solution obtained is not strictly boolean. Trees that are more favourable than others will receive a higher y_j value, but these values will still be less than 1. To make a final pick on the trees out of these y_j values, randomized rounding needs to be performed to round the values to either 1 or 0. For each net, the probability of one of the trees j being selected will be based on the y_j values. The bigger the y_j value, the higher the probability of being chosen. The randomized rounding algorithm is given in Figure 17.

Randomized Rounding Algorithm

```
For each net $i$
    If one of the tree values=1, then just select this tree and go to next net.
    $sum$=0
    $r$= generated random number between 1 and 0
    For each tree $j$ in net $i$
        $sum$+=$y_j$
        if ( $sum$>=$r$ ) then
            Pick tree $j$
            Go to next net.
        end
    End
End
```

Figure 17 - Randomized Rounding Algorithm.

3.6 Summary

This chapter explained how to model the routing problem as an optimization problem. Furthermore, past research on solving this formulation efficiently are discussed, including parallel solvers, Remove Constraint Optimization, and preprocessing.　The next chapter will look at two new optimizations, and how they can improve the solving time for the routing problem.

4 New Optimization Routing Model

4.1 Modified Model

The multi-objective dynamic Z_{max} model (*Model 2*) described in 3.1 was implemented for the purpose of this research. To translate the model in the form **Ay**<**b**, *Constraints 2.3*'s equality is transformed:

$$y_j \geq 0 \Rightarrow -y_j < 0$$

Since the problem benchmarks have the objectives in the minimization format, to utilize interior point method (which solves a maximization problem) one needs to convert the benchmarks from a minimization problem to a maximization problem. In the process of the conversion, we discovered that there are actually a few steps involved. Firstly, one needs to negate the weights in the objective function:

$$\text{Min} \sum w_j y_j \Rightarrow \text{Max} \sum (-w_j)\, y_j \quad \textit{(a)}$$

Second, we have to solve the relaxed the problem from IP to LP. Originally, since we want to select one tree, the *Constraint 2.1* was formulated as (these constraints are "active" constraints, the equality is same as inequality):

$$\sum y_j = 1 \Rightarrow \sum y_j \leq 1$$

The problem with this is that with the objective function in *(a)*, assuming that the overall coefficients for each y_j is negative, the algorithm will try to push all the y_j to zero, to achieve the maximum objective function. This is because $0 \leq y_j \leq 1$, and so $-\infty \leq$objective function≤ 0. In another words, to achieve the maximum objective function value the algorithm will push all the y_j to zero to get the most non-negative objective value. However, what we really want to do is to ensure that at least one tree is selected for each net. To enable this, we need to change *Constraint 2.1* in *Model 2* to:

$$\sum (-y_j) \leq -1 \quad \textit{(Constraint 2.1a)}$$

Doing so would prevent *all* of the trees to go to zero simultaneously, because with such constraint at least one x_j value has to be greater than one for each net.

Originally, as described in section 3.2, we calculate our initial point for interior point method by:

$$y^0_j < c_{min} / row_{max}, \text{ for all j}$$

The logic behind this is to find a $\mathbf{y^0}$ value such that $\mathbf{y^0}$ would not violate *Constraint 2.2*. (i.e. the sum of the congestion for any edges would not be over the maximum congestion allowed.) However, our implementation uses *Model 2*, where the Z_{max} is dynamically calculated. Thus, a value for c_{min} (minimum congestion limit) can not be obtained initially. More importantly, this method of initial point generation guarantees that $y^0 < 1$, because $row_{max} > c_{min}$. Having $\mathbf{y^0}<1$ could possibly violate our new constraint *Constraint 2.1a*. For instance, if $y_j^0=0.1$ for all j, then $\sum (-x_j) = -(0.1+0.1)=-0.2$ is not less than -1. This violates *Constraint 2.1a* and consequently will make interior point method unable to converge. Thus, the proposed new initial point generation method is:

$y^0_j = 1.1$, for all j
Z_{max} = max congestion caused by the initial point y^0

The $\mathbf{y^0}$ values will ensure that the initial point is a feasible solution subject to *Constraint 2.1a*, where as the Z_{max} value will ensure that *Constraint 2.2* would not be violated.

Finally, it should be mentioned that the y_j solution we obtained from solving the relaxed LP model can contain non-integer values. Although we expect most values to go to either 1 or 0, if there are trees that have equal weights, they will share the same non-integer value between zero and one. Thus, to convert back to an integer solution, a randomized rounding approach as specified in [4] is performed.

4.2 Remove Tree Optimization (RTO)

Using the same line of logic with the remove constraint optimization (RCO) described in section 3.4.1, another optimization approach is proposed here. The idea is that as we progress through the iterations, y_j is going to get closer and closer to the optimal solution. Thus, we don't expect a great variation in the values of y_j. So, once the y_j values have stabilized enough, the trees with small y_j values should be removed. This is summarized as follow:

Algorithm 3.2 version 1)

```
For each net, do {
   for each tree in current net, do {
       if ($y_j < 0.1$) {
              Remove:
                     o    the column in matrix A and row in vector y representing the tree
                     o    the row in matrix A and vector b that corresponds to constraint 3 of that tree.
       }
   }
}
```

In this case, vector **b** is the values of the right hand side of the constraints. Essentially, we are removing from the problem the trees that we think are going to be unselected. This will reduce problem size and reduce the iteration time. However, there is one problem with this formulation. If even the most preferable tree(s) has values less than 0.1, then we will remove all possible trees for the current net from the problem, and no trees will be chosen. This can occur in a scenario such as the following:

Ex. 3.2a)
Objective: Min $-y_1-y_2-\ldots.-y_{19}-y_{20}$
Subject to:
$-y_1-y_2-\ldots-y_{19}-y_{20}<-1$
$y_j>0$, for j = 1..20

In this case, since all the weights for the y_j are equal, all the trees are equally preferable. Since the sum of y_j can have a max value of -1 (subject to *Constraint 2.1a*), the trees will share this max value equally. This means that $y_j = 1/20 = 0.05$ for all *j*. This scenario gives rise to two problems:

1.) The tree with the max value for this net is less than 0.1.
2.) There are more than one tree that contains the max value for current net.

If we apply the remove tree optimization (RTO) to this problem, we will remove all possible trees for this net. To solve problem 1.), we need to check that the tree(s) with the max value for this net will not be removed. For problem 2.), since all the trees with the max value for current net are equally preferable, we will just randomly select the last tree in the net and remove all the other nets with the same max value. This step actually involves a bit of work. Since we are also removing the corresponding y_j values, *Constraint 2.1a* can be violated. For instance, in Ex 3.2a), say we are at the point when y_j=0.1 for all *j*. If we remove tree 1 to 19, we are left with y_{20} = 0.1, so *Constraint 2.1a* is violated, because -0.1 is not less than -1. Thus, what we want to do is that as we remove the trees, we should add its values to the tree that is going to be selected, so that *Constraint 2.1a* would not be violated. This give rise to the second version of *Algorithm 3.2*:

Algorithm 3.2 version 2)

```
For each net, do {
    **maxOfCurrentNet = max { $y_j$ | j for current net };**
    **if (more than 1 tree for this net has value=maxOfCurrentNet) {**
        **treeToSelect = (a tree with value == maxOfCurrentNet);**
        **for each tree, do{**
            **if (current tree<0.1 AND current tree ≠ treeToSelect) {**
                **Add current tree's $y_j$ value to treeToSelect's $y_j$;**
                **Set current tree's $y_j$ = 0;**
            **}**
        **}**
        **maxOfCurrentNet = $y_{treeToSelect}$;**
    **}**
    for each tree in current net, do {
        **if ($y_j$ < 0.1 AND $y_j$ ≠maxOfCurrentNet) {**
            Remove:
            o   the column in matrix A and row in vector y representing the tree
            o   the row in matrix A and vector b that corresponds to constraint 3 of that tree.
        }
    }
}
```

However, there is one problem with *Algorithm 3.2 version2*. Since we have increased the y_j value of the tree *treeToSelect*, we could violate *Constraint 2.2*. Consider the following scenario:

Ex.3.2b)

Objective: Min $-y_1-y_2-\ldots-y_{19}-y_{20}+Z_{max}$

Subject to:

$-y_1-y_2-\ldots-y_{19}-y_{20}<-1$

$-y_{20}-Z_{max}<0$

$y_j>0$, for j = 1..20

Suppose y_j has values of 0.1, and Z_{max}=0.2. When we remove tree j=1..19, we add their values to y_{20}. The updated values are y_{20}=1. In such case, the constraint $-y_{20}-Z_{max}<0$ is violated. Essentially, as we increase the value of the tree that is going to be selected, we need to increase the value of Z_{max} as well. The value of Z_{max} will be readjusted to the appropriate value at the subsequent iteration. This gives the version 3 of the RTO:

Algorithm 3.2 version 3)

```
For each net, do {
  maxOfCurrentNet = max { $y_j$ | j for current net };
     if there are more than 1 tree with value =maxOfCurrentNet for this net {
          treeToSelect = (a tree with value == maxOfCurrentNet);
          for each tree, do{
               if (current tree<0.1 AND current tree $\neq$ treeToSelect) {
                    Add current tree's $y_j$ value to treeToSelect's $y_j$;
                    **Add current tree's $y_j$ value to $Z_{max}$;**
                    Set current tree's $y_j = 0$;
               }
          }
          maxOfCurrentNet = $y_{treeToSelect}$;
     }
     for each tree in current net, do {
          if ($y_j < 0.1$ AND $y_j \neq$maxOfCurrentNet) {
               Remove:
               o    the column in matrix A and row in vector y representing the tree
               o    the row in matrix A and vector b that corresponds to constraint 3 of that tree.
          }
     }
}
```

So far, we have only considered removing the trees that are not going to be selected. A further optimization is to realize that if there's only one single tree that has a y_j value greater or equal to one, this means that tree is going to be selected. We can remove all the trees from consideration, and also remove the corresponding *Constraint 2.1a* from the problem. However, since we have selected a tree, we would need to subtract the congestion caused by the selected tree from the right hand side. This will ensure that our Z_{max} value took the selected tree's congestion into account. Lastly, if more than one tree has a value greater than or equal to one, then we don't apply this optimization, and just proceed with removing the unselected tree. This gives the final version of the RTO:

Algorithm 3.2 Remove Tree Optimization (RTO)

```
For each net, do {
  maxOfCurrentNet = max { $y_j$ | j for current net };
  removeAll = false;
  if there's one and only one tree that satisfy maxOfCurrentNet>=1 {
      removeAll=true;
  } else {
     if there are more than 1 tree with value=maxOfCurrentNet for this net {
          treeToSelect = (a tree with value == maxOfCurrentNet);
          for each tree, do{
               if (current tree<0.1 AND current tree $\neq$ treeToSelect) {
                    Add current tree's $y_j$ value to treeToSelect's $y_j$;
                    Add current tree's $y_j$ value to $Z_{max}$;
                    Set current tree's $y_j$ = 0;
               }
          }
          maxOfCurrentNet = $y_{treeToSelect}$;
     }
  }
  for each tree in current net, do {
     if (removeAll==true OR ($y_j$ < 0.1 AND $y_j \neq$maxOfCurrentNet) ) {
               Remove:
               o    the column in matrix A and row in vector y representing the tree
               o    the row in matrix A and vector b that corresponds to constraint 3 of that tree.
     }
  }
  if (removeAll==true) {
      Remove the row in matrix A and vector b that corresponds to constraint 1a.
      Subtract the congestion occupied from selected tree in vector b.
  }
}
```

Obviously, since we removed trees during the optimization, one would need to put back the removed y_j (with appropriate zeroes and ones value) back into vector **y** after the convergence of the algorithm. Second, the optimal *exitCondition* to apply RTO needs to be determined experimentally. If the RTO is applied too early, incorrect tree choices might be made because the solution might not have stabilized yet. If the RTO is applied too late, then unnecessary processing time will be wasted on finalizing choices that have been made already.

4.3 Required Changes in Randomized Rounding

In actual implementation, the randomized rounding algorithm needs to be modified to work properly. This is due to the nature of the linear relaxation from integer, and also the change in the constraints. The original algorithm described in section 3.5 is provided below for reference.

<u>Randomized Rounding Algorithm</u>

```
1.  For each net $i$
2.      If one of the tree values=1, then just select this tree and go to next net.
3.      $sum$=0
4.      $r$= generated random number between 1 and 0
5.      For each tree $j$ in net $i$
6.          $sum$+=$y_j$
7.          if ( $sum$>=$r$ ) then
8.              Pick tree $j$
9.              Go to next net.
10.         End
11.     End
12. End
```

The first required modification is to change the condition in step 2. In actual interior point computation, even when one exit as late as *exitCondition*=1e^{-6}, the trees that should get values of one never reach this value. Instead, the value might be a fractional value such as 0.99998, since the interior point iterations are approaching the actual solution with a margin of error. In order to take this into account, one can take a sufficiently accurate value of 0.9999, and step 2 should be changed to:

2. If one of the tree values>0.9999, then just select this tree and go to next net.

Second, since *constraint 1* is changed to:

$$\sum (-y_j) \leq -1 \quad \textit{(constraint 1a)}$$

y_j values can become greater than 1. Essentially, the y_j values are coming from the larger values and become smaller as the interior point iteration proceeds. Thus, step 4 needs to be modified to reflect the fact that the range for the sum of y_j values is not in the range {0,1} anymore. Step 4 should be changed to:

4. r= (generated random number between 1 and 0) * ($\sum_{y_{j \in N_k}} y_j$)

Essentially, instead of having the variable r range equal to {0, 1}, we are now having the range equal to {0, sum of all y_j that belong to the current net}. The modified version of the randomized rounding algorithm is given below.

Randomized Rounding Algorithm (Modified)

```
1.  For each net $i$
2.      If one of the tree values>0.9999, then just select this tree and go to next net.
3.      $sum$=0
4.      $r$= (generated random number between 1 and 0) * ( $\sum_{y_{j \in N_k}} y_j$ )
5.      For each tree $j$ in net $i$
6.          $sum$+=$y_j$
7.          if ( $sum$>=$r$ ) then
8.              Pick tree $j$
9.              Go to next net.
10.         End
11.     End
12. End
```

4.4 Parallel Solver for Projection Step

As described in section 3.2, the bottleneck in the interior point iteration is solving the linear system involving $\mathbf{A}^T\mathbf{D}^2\mathbf{A}$. This thesis also investigated the efficiency in using the *PSPASES* package described in section 3.3 to perform parallel solving on the linear system $(\mathbf{A}^T\mathbf{D}^2\mathbf{A})\ \mathbf{dx} = -\ \mathbf{c}$. The primary advantage of *PSPASES* package is that it is able to efficiently execute all four steps of direct solving in parallel. The details of the implementation can be found in [54]. Firstly, the fill reduced ordering is done using a parallel multilevel nested dissection algorithm [52], in which the problem is transformed to a graph and partitioned in a multilevel manner. Each partition is distributed to the different processors and ordered using the multiple minimum degree algorithm. The symbolic and numerical factorization phase uses a parallel multifrontal Cholesky factorization [53], where the graph is broken into multi parts using the elimination tree associated with the matrix **A** as a guide. Each part is then factored in different processors. Finally, the triangular solving is also done in parallel, using the elimination tree as a guide. [54]

The *PSPASES* package allows any MPI compatible library to be used for communication between different processors, and any BLAS library to be used for matrix and vector operations. In our implementation, the *GotoBLAS* library from Texas advanced computing center is utilized for performing vector and matrix operations, while the *MPICH2* library is used for MPI communication between different processors.

4.5 Summary

This chapter introduced the two new optimizations, the Remove Tree Optimization and the parallel solver. The required changes in the model and randomized rounding algorithm are also discussed. In the next chapter, the effects of these optimizations and their effects when combined with the optimizations discussed in Chapter 3 will be examined.

5 Experimental Results

In this chapter, the effects of the various optimizations discussed in chapter 3 and chapter 4 will be examined. Section 5.1 discusses the optimal exit condition found, whereas section 5.2 to 5.5 discusses the effects of applying RCO, RTO, simultaneous optimization, and the PSPASES parallel solver.

All of the experiments were executed in Windows Vista's MATLAB environment, on an Intel Core 6300 (Dual CPU) machine at 1.86GHz, with 3 gigabyte of memory. All of the coding is done in MATLAB, except the parallel solver *PSPASES*. Various real routing problems of different number of variables (number of trees) were benchmarked. All obtained solutions are randomly rounded using the method described in section 3.5 to finalize tree choices for each net.

5.1 Optimal Exit Condition

During the execution of the interior point method, the calculated solution proceeds closer and closer to the actual solution we are trying to achieve. From section 3.2, one can see that the exit point is determined at the condition *exitCondition*< *exitCriterion*, where *exitCondition* = $|\mathbf{c}^T\mathbf{x}^{k+1} - \mathbf{c}^T\mathbf{x}^k| / |\mathbf{c}^T\mathbf{x}^k|$. In other words, when the progress from one interior point loop to another is smaller than a certain exitCriterion, then it means that one got really close to the solution and it is appropriate to exit from the loop. An appropriate *exitCriterion* would allow for early exit (less execution time) and small error from the final answer (high accuracy).

To find the optimal exitCriterion when no optimization is applied, experiments were ran on four different problem sizes. The obtained objective function value from various exitCriterion is shown in Table 1. No optimization and preprocessing were used, and the obtained solution was randomized rounded at the end. To set a point of reference, the $1e^{-6}$ solution is treated as the final solution value. This is appropriate because each loop progression step is really small, and the solution is very close to the actual solution.

The percentage difference between solutions at various *exitCriterion* and the $1e^{-6}$ solution is shown in Table 2. As can be seen from the result, the percentage difference is insignificant starting at *exitCriterion* $1e^{-3}$ as the problem size increases. A safe optimal *exitCriterion* when no optimizations and preprocessing are applied is $1e^{-4}$.

Table 1. The obtained objective function value from various *exitCriterion*, for different problem sizes. No optimization and preprocessing, with randomized rounding at the end.

ProbSize	exitCriterion	1e-01	1e-02	1e-03	1e-04	1e-05	1e-06
262	Obj. Fn.	-19.38	-18.38	-18.27	-18.22	-18.30	-18.26
1670		-31.76	-28.90	-28.72	-28.72	-28.73	-28.73
2386		-32.20	-29.69	-29.48	-29.49	-29.48	-29.47
7241		-90.02	-84.66	-83.00	-83.00	-83.00	-83.00

Table 2. Percentage difference from 1e^{-6} answer, for various problem sizes.

% Difference from 1e^{-6} answer	*exitCriterion*					
ProbSize	1e-01	1e-02	1e-03	1e-04	1e-05	1e-06
262	6.17%	0.65%	0.08%	-0.23%	0.22%	0.00%
1670	10.58%	0.59%	-0.04%	-0.01%	0.00%	0.00%
2386	9.26%	0.74%	0.04%	0.07%	0.03%	0.00%
7241	8.45%	1.99%	0.00%	0.00%	0.00%	0.00%

5.2 Remove Constraint Optimization (RCO) Results

To avoid applying remove constraint optimization (RCO) too early and causes instability in the linear system, the RCO is applied starting from *exitCondition*=1e^{-3}. The results of applying RCO are verified by checking the objective function value in Table 3. As can be seen from the "Obj. Fn. Variation" row, the results are highly accurate. As well, as the problem size gets larger the timing improves with RCO. As shown in Table 4, the average time per loop required after the remove constraint is applied is reduced. This is because the problem size got smaller after the optimization is applied. For larger problems, the effect of the extra cost of optimization time is mitigated, because the time saved from subsequent loops dominates the time savings.

Table 3. Results of solving various problems with and without RCO.

Probem Size (# of variables)	262		1670		2386		7241	
Rmv Constr Opt Applied?	no	yes	no	yes	no	yes	no	yes
# of Iterations	17	17	22	22	21	21	30	30
Time executed	0.38	0.41	4.56	4.73	6.12	6.44	160.65	154.89
Obj. Fn. Value	-18.30	-18.24	-28.73	-28.73	-29.48	-29.47	-83.00	-83.00
Time Improvement		-7.27%		-3.85%		-5.15%		3.59%
Obj. Fn. Variation		-0.32%		0.00%		-0.02%		0.00%

Table 4. Average time per loop before and after RCO.

Probem Size (# of variables)	**262**	**1670**	**2386**	**7241**
Avg time/loop before remove constr opt	0.02	0.19	0.27	5.19
Avg time/loop after remove constr opt	0.02	0.15	0.23	2.85
% of time reduced	-4.55%	31.03%	17.39%	82.11%

5.3 Remove Tree Optimization (RTO) Results

As discussed in section 4.2, the optimal *exitCondition* to apply RTO needs to be determined experimentally. It was determined that the optimal point to apply the optimization is when *exitCondition*=1e^{-3}. Due to the result found in section 5.4, the *exitCriterion* is set at 1e^{-4} when generating these results. The number of trees that was removed is shown in Table 5. Note that the amount of trees that has been removed is over 48% for all problem sizes. This is a significant amount of reduction, and it indicates that only a small percentage of tree choices need to be finalized at this point. As can be seen from the timing results in Table 6, timing improved as the problem size gets larger. Even though the computation in the RTO consumes extra processing time, the reduced loop time saved after the RTO is applied dominates. For instance, as can be seen from Table 7, for problem 7241 the average time per loop is reduced by a factor of 106, due to a significant reduction in the problem size. However, it would require a problem almost double the size (13265), to counter balance the RTO computation time and obtain a speedup of 32.74%.

Table 5. Number of trees removed with RTO, for different problem sizes.

ProbSize	**Number of Trees Removed**
262	233
1670	1434
2386	1922
7241	6799
13265	6353

Table 6. Number of loops and time required to solve with and without RTO.

Probem Size (# of variables)	262		1670		2386		7241		13265	
Rmv Tree Opt Applied?	no	yes	no	yes	no	yes	no	yes	no	yes
# of Iterations	13	21	19	26	17	27	26	31	26	36
Time executed	0.24	0.52	3.93	5.62	5.01	8.89	136.27	177.31	476.70	320.63
Obj. Fn. Value	-18.26	-18.64	-28.72	-29.69	-29.48	-30.13	-83.00	-84.00	-66.12	-66.75
Time Improvement		-119.22%		-43.00%		-77.45%		-30.12%		32.74%
Obj. Fn. Variation		2.08%		3.38%		2.20%		1.20%		0.95%

Table 7. Comparison of loop time before and after RTO.

Probem Size (# of variables)	262	1670	2386	7241	13265
Avg time/loop before remove tree opt	0.02	0.19	0.27	5.04	4.78
Avg time/loop after remove tree opt	0.03	0.03	0.04	0.05	0.48
% of time reduced	-35.00%	540.00%	677.14%	10623.40%	895.83%

5.4 Simultaneous Optimization

The objective values obtained from applying simultaneous remove tree, RCO, and preprocessing on different problem sizes are shown in Table 8, for exit conditions from 1e^{-1} to 1e^{-6}. Randomized rounding was used at the end. From the results of section 5.2 and 5.3, the remove constraint and RTO is applied at condition=1e^{-3}. The percentage difference of the resulting objective function value from the 1e^{-6} objective function value is shown in Table 9. As can be seen from the table, the exit condition in which zero error from the 1e^{-6} value is achieved is at 1e^{-4}. This indicates that the *exitCriterion* can be set at 1e^{-4}.

Table 8. Objective function value for different problem sizes, from exitCondition 1e^{-1} to 1e^{-6}.

ProbSize	exitCondition	With Optimzation?	1e-01	1e-02	1e-03	1e-04	1e-05	1e-06
262	Obj. Fn.	N	-19.23	-18.31	-18.30	-18.62	-18.62	-18.62
		Y	-19.38	-18.38	-18.27	-18.22	-18.30	-18.26
1670		N	-29.99	-28.87	-28.73	-29.69	-29.69	-29.69
		Y	-31.76	-28.90	-28.72	-28.72	-28.73	-28.73
2386		N	-30.92	-29.57	-29.49	-30.13	-30.13	-30.13
		Y	-32.20	-29.69	-29.48	-29.49	-29.48	-29.47
7241		N	-86.82	-84.31	-83.00	-84.00	-84.00	-84.00
		Y	-90.02	-84.66	-83.00	-83.00	-83.00	-83.00

Table 9. Percentage difference in objective function, wit exitCondition=1e^{-6} as reference.

% Difference from 1e-6 value	exitCondition					
ProbSize	1e-01	1e-02	1e-03	1e-04	1e-05	1e-06
262	3.26%	-1.68%	-1.72%	0.00%	0.00%	0.00%
1670	1.02%	-2.76%	-3.24%	0.00%	0.00%	0.00%
2386	2.64%	-1.85%	-2.12%	0.00%	0.00%	0.00%
7241	3.35%	0.36%	-1.19%	0.00%	0.00%	0.00%

The number of loops and the time required to solve different problems are given in Table 10. From the data it can be seen that the required number of loops increase when the optimizations are applied. This is because when the matrix **A** is modified by removing constraints and trees, the left-over y_j now contributes a greater portion on the *exitCondition*, and thus the interior point method needs time to arrive at the same *exitCondition* again. Consequently, it would require more iterations to settle down. However, since the matrix size is now significantly smaller, each iteration requires much less time to complete. Thus, as can be seen from Table 11, even though the number of loops increases, the time required to obtain a solution is decreasing as the problem gets larger. For the tested problems, in the case of exitCondition=1e^{-4} the solving time can reduce by up to 48%. This is because a significant number of rows and columns are removed from the matrix **A**. As shown in Table 12, the total number of inactive constraints, unchosen trees, singleton trees, and unneeded constraints contributes to a big portion of the original matrix **A**. In the case of problem size 7241, the total number of rows reduced is near 75% of the original matrix. In addition, in RTO and remove singleton trees portion of the preprocessing operation, columns of matrix **A** are reduced as well. This further reduces the size of matrix **A** and thus the interior point loop time.

Table 10. Number of loops and solving time, for different exitCondition and problem sizes.

ProbSize	With Optimzation?	exitCondition	1e-04	1e-05	1e-06
262	N	nCounts	13	15	17
		t (sec.)	0.29	0.34	0.38
	Y	nCounts	24	26	27
		t (sec.)	0.68	0.75	0.78
1670	N	nCounts	19	21	22
		t (sec.)	3.76	4.14	4.34
	Y	nCounts	26	29	31
		t (sec.)	3.19	3.32	3.41
2386	N	nCounts	17	19	21
		t (sec.)	4.94	5.48	6.02
	Y	nCounts	28	30	32
		t (sec.)	3.58	3.68	3.79
7241	N	nCounts	26	28	30
		t (sec.)	134.23	144.18	154.14
	Y	nCounts	33	34	39
		t (sec.)	113.46	113.53	113.90
13265	N	nCounts	36	43	49
		t (sec.)	478.37	566.80	655.38
	Y	nCounts	40	42	44
		t (sec.)	246.60	251.60	252.31

Table 11. Percentage difference in number of loops and solving time, relative to no optimization.

ProbSize	exitCondition	1e-04	1e-05	1e-06
262	nCounts	84.62%	73.33%	58.82%
	t (sec.)	132.60%	121.93%	105.25%
1670	nCounts	36.84%	38.10%	40.91%
	t (sec.)	-15.24%	-19.75%	-21.43%
2386	nCounts	64.71%	57.89%	52.38%
	t (sec.)	-27.51%	-32.75%	-37.06%
7241	nCounts	26.92%	21.43%	30.00%
	t (sec.)	-15.47%	-21.26%	-26.11%
13265	nCounts	11.11%	-2.33%	-10.20%
	t (sec.)	-48.45%	-55.61%	-61.50%

Table 12. Number of constraints removed, and percentage relative to original number of rows.

ProbSize	Number of rows in Matrix A	01exitConditions	1E-04	% of rows removed
262	478	ConstrRemoved	35	7.32%
		TreesRemoved	169	35.36%
		SingletonTreesRemoved	61	12.76%
		UnneededConstrRemoved	32	6.69%
1670	3013	ConstrRemoved	237	7.87%
		TreesRemoved	1037	34.42%
		SingletonTreesRemoved	397	13.18%
		UnneededConstrRemoved	342	11.35%
2386	4504	ConstrRemoved	320	7.10%
		TreesRemoved	1063	23.60%
		SingletonTreesRemoved	865	19.21%
		UnneededConstrRemoved	343	7.62%
7241	10779	ConstrRemoved	424	3.93%
		TreesRemoved	5500	51.03%
		SingletonTreesRemoved	1299	12.05%
		UnneededConstrRemoved	815	7.56%
13265	26845	ConstrRemoved	1079	4.01%
		TreesRemoved	5831	21.72%
		SingletonTreesRemoved	4336	16.15%
		UnneededConstrRemoved	6394	23.82%

5.5 Using *PSPASES*

The *PSPASES* package, as described in 4.4, was used to solve the linear system involving $\mathbf{A}^T\mathbf{D}^2\mathbf{A}$ of various sizes. The solving of the system consists of four steps: ordering, symbolic factorization, numerical factorization, and triangular solve. The performance in solving each of the systems is listed in Table 13. As can be seen from the table, the ordering and symbolic factorization time always improves as one switches from 2 to 4 processors. However, for numerical factorization phase, only the smallest and largest problem has improved performance. Since the running time for the smallest problem is negligible, it can be concluded that to benefit from parallel computing, the problem size needs to be large to overcome the cost of communication overhead. For the tested problems, the triangular solve phase receives no benefits from the additional 2 processors. However, from Table 14, one can see that for the smallest and biggest problem, the total solving time has improved by over 10%. The total solving time is dominated by the ordering and numerical factorization phase. Thus, once both of them are improved, the total solving time will be improved as well. This shows that the parallel solving approach improves the solving time, which confirms the research results of other parallel interior point methods that use different methodology for different problems. For example, in [55], [56], and [57], parallel interior point solution methods have all shown to provide speedup, especially when the percentage of computations that can be done in parallel is high.

Table 13. Execution time for different phase during *PSPASES* solve.

Order time (s)		
Processors	2	4
Problem		
2382	0.134	0.133
6183	4.233	2.896
7241	3.738	2.732
15704	8.961	6.466

Num. Factor time (s)		
Processors	2	4
Problem		
2382	0.050	0.030
6183	10.342	12.234
7241	11.007	14.546
15704	32.037	30.036

Symbolic time (s)		
Processors	2	4
Problem		
2382	0.019	0.014
6183	0.587	0.350
7241	0.490	0.341
15704	1.144	0.712

Triangular Solve time (s)		
Processors	2	4
Problem		
2382	0.005	0.006
6183	0.147	0.215
7241	0.233	0.272
15704	0.533	0.638

Table 14. Total execution time for *PSPASES* solve.

Total Solve time (s)			
Problem	2	4	% Improved
2382	0.208	0.183	12.02%
6183	15.309	15.695	-2.52%
7241	15.468	17.891	-15.66%
15704	42.675	37.852	11.30%

5.6 Summary

In this section, the effects of the RCO, RTO, simultaneous optimization, and parallel solving are discussed. It can be seen that these methodologies provide efficient speedup. In the next chapter, the conclusions and future work will be presented.

6 Conclusions and Future Work

Normal sequential routing approach does not guarantee a global optimal solution. An integer programming approach is able to incorporate multiple conflicting design objectives, and achieve globally optimal solution. One major issue in the integer programming approach is to speed up the solving process for more efficient routing. This research proposed several ways one can optimize the solving time using interior point method. The results demonstrated that the proposed optimization approaches provide acceleration and reduces the solving time. In section 6.1, the effects of the optimizations will be concluded, and their limitations, scalability, and contribution will be discussed. A discussion of future work is presented in section 6.2.

6.1 Effectiveness of Optimizations

Previous approach of optimizations, namely the preprocessing technique from [7] and the RCO technique from [2] are applied to speedup solving the routing problem using interior point method. Using a property derived from [7], constraints that will never be violated are removed from the problem to reduce the problem size. Furthermore, after the interior point loop has stabilized, the constraints that are unlikely to be fulfilled are removed as well.

A further problem-downsizing approach is proposed in this research. After the interior point method has progressed enough, solutions that have stabilized can be removed from the problem. This allows a great number of trees and nets to be eliminated from the problem, thereby speeding up the subsequent iterations. In the small routing problems, the timing is worsened by using this optimization, because the cost to decide what nets and trees to remove is greater than the time saved by the subsequent iteration. However, it was shown that for large problems, the timing can improve by as much as 32.74%, with only 0.95% difference in the objective function value.

Combined optimizations of preprocessing, RCO, and RTO has also been shown to be effective. From the experiments, it was shown that there was no instability in applying the remove constraint and RTO simultaneously. Furthermore, the timing can be improved by as much as 48% for large problems. Up to three quarters of the problem size can be reduced when the optimization is applied, which accelerates the solving time greatly.

The downsizing of the problem not only allowed for time saving, but it can also help in memory consumption as well. Furthermore, if the solver is run on a machine with limited memory, significant time savings can be achieved by avoiding a constant swap in page files.

For the RTO optimization, the amount of time improvement will depend on the nature of the problem. The more trees there are for each net, the bigger the problem size. In such case, RTO will perform particularly well, as it will be able to remove a large number of trees. However, note that for the tested problems, the ratio of the number of nets to the number of trees is quite low already. Thus, it is expected that the RTO will almost always guarantee improved results.

The key timing bottle neck in the interior point method is to solve the linear system involving $\mathbf{A}^T\mathbf{D}^2\mathbf{A}$. This research investigated the effects of utilizing an efficient parallel solver package to speedup the solving time. By utilizing the efficient *GotoBLAS* and *MPICH2* library along with *PSPASES*, the solving time can be improved due to parallel solving. It was found that for smaller problems, the communication cost dominates and no speedup was obtained. However, for bigger problems, up to 11.3% speedup can be achieved by using four processors instead of two. When the problem sizes gets bigger and bigger, utilizing even more processors should allow for even greater speedup.

For the parallel solving optimization, the amount of speedup will depend on the size of the problem and the number of processors utilized and memory size available. Furthermore, it will depend on the computer architecture that the solver is running on. A shared-memory machine will have a much better performance than a distributed system, as the communication cost during data distribution will be significantly lower. As well, if the problem size is too small, then using multiple CPUs will actually has an adverse effect on the solving time. This is because the increase in the communication time will be greater than the improvement from performing parallel computations. However, for larger problems and assuming a shared-memory system is used, it is expected that speedup will always be obtained. As the problem gets larger, higher speedup will be obtained due to ability to utilize higher number of CPUs. It is expected that the speedup will reach a limit when one reaches the capacity of the shared-memory, and swapping to disk will have to be performed. In such case, this optimization will be significantly slowed down, and one solution will be using a distributed array of shared-memory system.

It is difficult to compare the results of this optimized routing method compared to other people's research, because of the speed, memory, and architecture variance in the testing computers. However, the important thing from the research is not the absolute running speed, but the speedup one can obtain from performing these optimizations. Regardless of the implementations, the speedup from the optimization will always be there. The main impact from this research is in several ways. Firstly, a new RTO optimization is proposed, which allows for significant reduction in the problem size and solving time speedup. Secondly, the combined effects of the RTO optimization and other previous optimization techniques are shown and have proven to be effective in reducing the solving time. Lastly, the bottleneck of the interior point method is parallelized solved, and it is shown that this method is effective in obtaining speedup in solving routing problems. These three areas have not been researched before for routing, and it provides a valuable insight into how the routing problem can be solved more efficiently, when modeled as an optimization problem. Since the number of transistors is increasing greatly in VLSI design, these optimization techniques will be useful in obtaining a multiple constrained globally optimal routing solution in reasonable time.

6.2 Future Work

The work completed in this research can be enhanced to further speedup the solver. For instance, there are various standard preprocessing techniques to reduce the size of the matrix **A** before the interior point method is applied. Obviously, if the structure of the matrix is changed, further investigation would be required to understand how the RCO and RTO optimization need to be modified.

Problems with up to 10^5 variables were solved in this research. It would be interesting to see the effects of the optimization on bigger problems, such as problems with 10^6 or more variables. However, the results of the experiment prove that the optimization is scalable. The RTO optimization actually gives better speedup as the problem gets larger. Also, the effectiveness of the RTO optimization is independent of the problem size, as long as the problem is greater than a certain size. Moreover, the speedup of the parallel solving technique increases as the problem size increase. This implies that these two optimizations are scalable to the routing problem size.

References

[1] “Moore's Law: Made real by Intel® innovation”, Nov, 2006. http://www.intel.com/technology/mooreslaw/index.htm.

[2] A. Vannelli, “An Adaptation of the Interior Point Method for Solving the Global Routing Problem”, IEEE trans on Computer-Aided Design, vol. 10, pp. 193-203, Feb. 1991.

[3] L.Behjat, D.Kucar, A.Vannelli, “A Novel Eigenvector Technique for Large Scale Combinatorial Problems in VLSI Layout”, Journal of Combinatorial Optimization, Vol 6, no.3, pp.271-286, 2002.

[4] Z.Yang, “Hierarchical Global Routing with Multi-Objectives”, PhD proposal, Dept Electrical and Computer Engineering, University of Waterloo, 2005.

[5] C.Luo, “Novel Convex Optimization Approaches for VLSI Floorplanning”, PhD thesis, Dept Electrical and Computer Engineering, Unversity of Waterloo, 2008.

[6] L.Behjat, “New modeling and optimization techniques for global routing problem”, PhD thesis, Dept Electrical and Computer Engineering, University of Waterloo, 2002.

[7] W.L.Hare, M.J.J. Liu, T.Terlaky, “Efficient preprocessing for VLSI optimization problems”, Computational Optimization and Applications, Springer Netherlands, March 15, 2008.

[8] Mahesh Joshi, “PSPASES Home Page”, Nov, 2008. http://www-users.cs.umn.edu/~mjoshi/pspases/

[9] C. Y. Lee, “An Algorithm for Path Connection and its Application", IRE Transactions on Electronic Computers, vol. 10, pp. 346-365, 1961.

[10] S. M. Sait and H. Youssef, VLSI Physical Design Automation, McGraw-Hill, London, England, 1995.

[11] J. Soukup, “Fast Maze Router", In Proceedings of the Design Automa-tion Conference, pp. 100-102, 1978.

[12] F. Hadlock, “Finding a Maximum Cut of a Planar Graph in Polynomial Time", SIAM Journal of Computing, vol. 11, pp. 885-892, 1975.

[13] D. W. Hightower, “A solution to the line routing problem on a continous plane", In Proceedings of 6th Design Automation Workshop, pp. , 1969.

[14] K. Mikami and K. Tabuchi, “A computer program for optimal routing of printed circuit connectors", In Proceedings of IFIPS Conf., pp. 1475-1478, 1968.

[15] E.S. Kuh and M. Marek-Sadowska, Global Routing, volume 1, pages 133-168. Elsevier Science Publishers B.B., Amsterdam, Netherlands, 1985.

[16] C.J.Alpert, M. Hrkic, J.Hu, A.B. Kahng, J.Lillis, B. Liu, S.T. Quay, S.S. Sapatnekar, A.J. Sullivan, and P. Villarrubia. “Buffered Steiner Trees for diffcult Instances”, In Proceedings of International Symposium on Physical Design, pages4-9, 2001.

[17] S.Areibi, M.Xie, and A. Vannelli, “An efficient Steiner Tree Algorithm for VLSI Global Routing”, In proceedings of Canadian Conference on Electrical and Computer Engineering, volume 2, pages 1067-1072, 2001

[18] C.Chiang, C.K. Wong, and M. Sarrafzadeh, “A Weighted Steiner Tree-based Global Router with Simultaneous Length and Density Minimization.”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13:1461-1469, 1994

[19] J. Cong, A.B. Kahng, and K. Leung, "Efficient Algorithms for Minimum Shortest Path Steiner Arborescence Problem with Application to VLSI Physical Design", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 17(1):24-39, 1998.

[20] J. Cong and P.H. Madden, "Performance Driven Routing with Multiple Sources", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 16:410-419, 1997.

[21] J. Hu and S.S. Sapatnekar, "A Timing-Constrained Simultaneous Global Routing Algorithm", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 21:1025-1036, 2002

[22] R. Kastner, E. Bozorgzadeh, and M. Sarrafzadeh. "An Exact Algorithm for Coupling-Free Routing", International Symposium on Physical Design, pages 10-15, 2001

[23] W.A. Dees and P.G. Karger, "Automated Rip-Up and Reroute Techniques", 19th Design Automation Conference, pages 432-439, 1982.

[24] R. T. Hadsell and P. H. Madden, "Improved Global Routing through Congestion Estimation", In Proceedings of the Design Automation Conference, pp. 28-34, IEEE/ACM, Anaheim, CA, 2003.

[25] M.P. Vecchi and A. Kirkpatrick, "Global Wiring by Simulated Annealing", IEEE Transactions on Computer-Aided Design, 2:215-222, 1983.

[26] G. B. Dantzig, "Linear Programming and Extensions," Princeton, NJ:Princeton University, 1963.

[27] T.C. Hu and M.T. Shing, "A DecompositionAlgorithm for Circuit Routing", pages 144-152, IEEE press, New York, 1985.

[28] A.P. Ng, P. Raghavan, and C.D. Thompson, "Experimental Results for a Linear Program Global Router", Computer Artificial Intelligence, 6:130-143, 1987.

[29] P. Raghavan, "Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs", Journal of Computer Science, 37:130-143, 1988.

[30] P.Raghavan and C. D. Thompson, "Multi-Terminal Global Routing: A Deterministic Approximation Scheme", Algorithmica, 6:73-82, 1991.

[31] W. K. Luk, P. Sipal, M. Tamminen, D. Tang, L. S. Wong, and C. K. Wong, "A hierarchical global wiring algorithm for custom chip design", IEEE Transactions Computer-Aided Design, vol. 6, pp. 518-533, 1987.

[32] M. Marek-Sadowska, "Global router for gate array", In Proceedings of the IEEE International Conference on Computer Aided Design, pp.332-337, 1984.

[33] J. Cong, J. Fang, and Y. Zhang, "Multilevel Approach to Full-Chip Gridless Routing", In Proceedings of the 2001 International Conference on Computer Aided Design, pp. 396-403, 2001.

[34] M. Hanan, "On Steiner's Problem with Rectilinear Distance", SIAM Journal of Applied Mathematics, vol. 14, pp. 255-265, 1966.

[35] M. R. Garey and D. S. Johnson, "The rectilinear steiner tree problem is np-complete", SIAM Journal of Applied Mathematics, vol. 32, pp.826-834, 1977.

[36] J.B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem", In Proceedings of the American Mathematical Society", volume 7, pages 48-50, 1956.

[37] R. Tarjan, "Data Structures and Network Algorithms", Society for Industrial and Applied Mathematics, 1983.

[38] F. K. Hwang, "On Steiner Minimal Trees with Rectilinear Distance", SIAM Journal of Applied Mathematics, vol. 30, pp. 104-114, January 1976.

[39] N. Karmarkar, “A new polynomial-time algorithm for linear programming,”, Combinatorica, vol. 4, no.4, pp. 373-395, 1984.

[40] R.J. Vanderbei, M.S. Meketon, and B.A. Freedman, “A Modification of Karmarkar’s Linear Programming Algorithm”, Algorithmica, 1:395-407, 1986.

[41] E.R. Barnes, “A Variation on Karmarkar’s Algorithm for Solving Linear Programming Problems”, Mathematical Programming, 36:174-182, 1986.

[42] I. Adler, N. Karmarkar, M.G.C. Resende, and G.. Veiga, “Data Structures and Programming Techniques for the Implementation of Karmarkar’s Algorithm”, ORSA Journal on Computing, 1:84-106, 1989.

[43] M. Kojima, S. Mizuno, and A. Yoshise, “A Primal-Dual Interior-Point Method for Linear Programming, pages 29-47”, Springer-Verlag, New York, 1987.

[44] S. Mehrotra, “On the Implementation of a Primal-Dual Interior Point Method”, SIAM Journal of Optimization, 2(4):575-601, 1992.

[45] A.V. Fiacco and G.P. McCormick, “Nonlinear Programming: Sequential Unconstrained Minimization Techniques”, John Wiley and Sons, New York, 1968.

[46] I. Adler, N. Karmarkar, M. G. C. Resende, and G. Veiga, “An implementation of Karmarkar’s algorithm for linear programming”, *Math. Program.,* vol. 44, pp. 297-335, 1989.

[47] Anshul Gupta, Vipin Kumar, and A. H. Sameh., “Performance and scalability of preconditioned conjugate gradient methods on parallel computers”, IEEE Transactions on Parallel and Distributed Systems, 1995.

[48] I. C. Decker, D. Falcão, and E.Kaszkurewicz, “Conjugate gradient methods for power system dynamic simulation on parallel computers,” IEEE Trans. Power Systems, vol. 11, pp. 1218-1227, Apr. 1996.

[49] E. F. D’Azevedo, V. L. Eijkhout, and C. H. Romaine. Conjugate gradient algorithms with reduced synchronization overhead on distributed memory multiprocessors. Technical Report 56, Lapack Working Note, August 2002.

[50] A. Jordan, R. P. Bycul, “The Parallel Algorithm of Conjugate Gradient Method”, Lecture Notes on Computer Science, Vol. 2326, pp. 156-165, 2002.

[51] R. Bycul, A. Jordan, M. Cichomski, “A new version of conjugate gradient method parallel implementation”, Proceedings of the International Conference on Parallel Computing in Electrical Engineering (PARELEC), Warsaw, 2002, pp. 318–322.

[52] G. Karypis, V. Kumar, “Parallel algorithm for multilevel graph partitioning and sparse matrix ordering”, Journal of parallel and distributed computing, vol.48, pp.71-85, 1998.

[53] A. Gupta, G. Karypis, V. Kumar, “Highly scalable parallel algorithms for sparse matrix factorization”, IEEE Transactions on Parallel and Distributed Systems”, vol.8, no.5, pp.502-520, 1997.

[54] Gupta, A., Gustavson, F., Joshi, M., Karypis, G., and Kumar, V., “*PSPASES*: an efficient and scalable parallel sparse direct solver”, In Kluwer Intl. Series in Engineering and Science, T. Yang, Ed. Vol. 515. Kluwer, 1999.

[55] De Silva, A. and Abramson, D. A, "A Parallel Interior Point Method and its Application to Facility Location Problems", Computational Optimization and Applications, Volume 9, Number 3, March 98, pp 249 – 273

[56] G. Karypis, A. Gupta and V. Kumar, "Parallel Formulation of Interior Point Algorithms," Technical Report 94-20, Dept. of Computer Science, Univ. of Minnesota, Apr. 1994.

[57] R. Levkovitz, J. Anderson, and G. Mitra, "Interior point method for LP on parallel computers," in System modelling and Optimization, Proc. of the 15th IFIP conf., Zurich, Switzerland, 1991.