Timing Driven Global Routing for Standard Cell Design

by

Amir Hashmi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

June, 1995

Timing Driven Global Routing for Standard Cell Design

BY

Amir Hashmi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

In

Computer Science

June 1995

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

DHAHRAN, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by

Amir Hashmi

under the direction of his Thesis Advisor, and approved by his Thesis committee, has been presented to and accepted by the Dean, College of Graduate Studies, in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Thesis Committee :

[illegible]

Dr. Talal H. Maghrabi (Chairman)

[illegible]

Dr. Habib Youssef (Co-Chairman)

[illegible] [illegible]

Dr. Mohammed Al-Suwaiyel (Member)

Sadiq Sait. M.

Dr. Sadiq M. Sait (Member)

[illegible]

Dr. Muhammed Al-Mulhem
Department Chairman

[illegible]

Dr. Ala H. Rabeh
Dean, College of Graduate Studies

Date: 1/7/95

COLLEGE OF GRADUATE [illegible]

Timing Driven Global Routing for Standard Cell Design

MS Thesis

Amir Hashmi

June, 1995

Dedicated to

my parents

&

my brother and sisters

whose prayers, guidance, and inspiration led to

this accomplishment

Acknowledgment

In the name of Allah, Most Gracious, Most Merciful. All praise to Allah, *subhanahu-wa-ta'ala*, the Almighty, for all his blessings on me. I feel priviledged to glorify His name in the sincerest way through this small accomplishment. I seek His mercy, favor, and forgiveness. And I ask Him to accept my little effort.

Acknowledgement is due to King Fahd University of Petroleum and Minerals for providing support to this research work.

I would like to express my deep gratitude to my thesis committee chairman, Dr. Talal H. Maghrabi for his assistance and support. I am indebted to my thesis co-chairman Dr. Habib Youssef for his time, guidance and suggestions. I am also grateful to my committee members Dr. Sadiq M. Sait and Dr. Mohammad Al-Suwaiyel for their sincere help and advice.

I am thankful to the chairman of department of Information and Computer Scdience, Dr. Muhammed Al-Mulhem and other faculty members for their cooperation.

I express my gratitude to my friends Shahid Tanvir, Tambi Baik and Khalid Al-Farra for their constant help, guidance and moral support during this work. I am thankful to my fellow graduate students and all my friends on the campus especially Asim, Nayyar and Shahid Ali who provided a wonderful company throughout my stay at KFUPM.

Special thanks go to my fellow and friend late Emad Amin Khan for his moral support and encouragement. May Allah bless his soul.

Last but not the least, thanks are due to the members of my family for their emotional and moral support throughout my academic career. This work is dedicated to my family for taking pains to fulfill my academic pursuits and building my personality.

Contents

List of Tables

List of Figures

Abstract

Name:	**Amir Hashmi**
Title:	**Timing-driven Global Routing for Standard Cell Design**
Major Field:	**Computer Science**
Date of Degree:	**June, 1995**

In the VLSI design process, interconnect delays play an important role in determining the performance of the circuits as they can make it impossible to achieve the required clock rate. Nowadays, it is rare to find a placement program that does not take into consideration timing issues of the circuit. However, routing did not receive similar attention. We believe that timing of the layout can be further improved if timing critical nets are given preferential treatment during routing. Such an approach has been taken in this work. This thesis accomplishes the implementation of a timing-driven global router program for standard cell VLSI design. The solution quality has been measured in terms of path delays, interconnection length and layout area. An iterative improvement technique called Tabu Search has been used to improve the initial global routing solution. This technique has been compared with another technique called Simulated Annealing. Tabu search has resulted in better solutions with less execution times in all the test cases used.

Master of Science Degree
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
June 1995

خلاصة الرسالة

الإسم	:	عامر هاشمى
عنوان الرسالة	:	تسليك الدوائر المحكوم بالزمن للتصاميم ذات الخلايا القياسية
التخصص	:	علوم الحاسب الالى
تاريخ الشهادة	:	حزيران / يونيو ١٩٩٥ م

تلعب الإعاقات الزمنية بين التوصيلات المختلفة في عملية تصميم الدوائر المتكاملة ذات النطاق الواسع جدا دورا مهما في تحديد أداء الدوائر، اذ أنه من الممكن أن تؤدي هذه الإعاقات إلى استحالة الحصول على السرعة المطلوبة للساعة. و من النادر في عالمنا اليوم أن يوجد برنامج لتوضيع الدوائر لا يأخذ بعين الإعتبار عنصر الزمن للدوائر. ولكن الحال ليست كذلك بالنسبة لتسليك الدوائر. فنحن نعتقد بأن مزامنة مخطط ما للدوائر يمكن أن تتحسن إذا تم تسليك أسلاك الوصل الحرجة أولا. و لقد إتبع هذا الاسلوب في هذا العمل. في هذه الرسالة يتم تنفيذ برنامج تسليك للدوائر محكوم بالمزامنة لتصميم الدوائر المتكاملة ذات النطاق الواسع جدا وذات الخلايا القياسية. و يتم قياس نوعية الحل بواسطة مسارات الإعاقات الزمنية, طول التوصيلات, و مساهمة مخطط الدوائر. وقد إستخدم أسلوب تحسيني متكرر يسمى طريقة (Tabu Search) لتحسين الحل الأولي لعملية تسليك الدوائر. وقد قورن هذا الأسلوب باسلوب آخر يسمى طريقة (Simulated Annealing).و قد نتج عن طريقة (Tabu Search) حلول أفضل مع أزمنة تنفيذ أصغر في جميع حالات الإختبار التي استخدمت.

درجة الماجستير فى العلوم

جامعةالملك فهد للبترول والمعادن
الظهران – المملكةالعربيةالسعودية

حزيران / يونيو ١٩٩٥ م

Chapter 1

Introduction

1.1 Overview

In the automated design of VLSI circuits, the two principal design steps are *placement* and *routing.* The placement step involves the assignment of cells of the circuit to fixed locations of the chip, while routing is the task of finding suitable paths to interconnect the desired set of pins of the cells. A set of pins or terminals is called a *net* with one terminal designated as a source and the rest as sinks. The term *suitable path* is meant for those paths that minimize a given objective function, subject to constraints. Constraints may include number of routing layers, minimum separation between adjacent wires of different nets, minimum width of routing wires, timing, etc. Objective functions include the reduction in the overall required wirelength and avoidance of timing problem due to delays of interconnection.

Routing takes a major portion of design time (about 30%) and as much as 50% of layout area. In order to achieve routing for generally large number of interconnected cells, computer programs called *routers* are used which precisely define the electrical paths among the pins of the interconnected cells.

The prime objective of a routing algorithm is to achieve circuit interconnection in the minimum possible area, automatically, and, with minimum manual intervention. Thus most routers are meant to achieve complete automatic routing using the smallest possible wire length to satisfy the performance criteria. For instance, consider Figure 1.1(a) in which a shortest path is shown with length equal to the Manhattan distance or the rectilinear distance between a and b. Manhattan distance between two points p and q is given by

$$dist(p,q) = \mid p_x - q_x \mid + \mid p_y - q_y \mid \tag{1.1}$$

The path in Figure 1.1(b) has more bends and is longer. Thus a signal will take longer time to travel on this path.

The routing process basically includes two major steps: *global routing* and *detailed routing.* Global routing is performed first to achieve a routing plan for the nets such that they can be assigned to particular regions. Moreover, to optimize the routing plan some additional connection points are determined to get connection among various routing areas (channels). In detailed or channel routing, each routing region is picked up and nets are assigned to particular tracks in that region

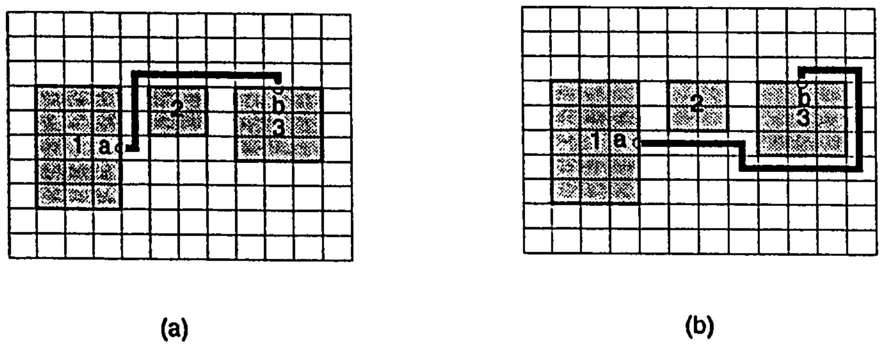


Figure 1.1: Two possible paths connecting a pair of points. (a) The shortest path. (b) A longer path with more bends [1].

to achieve the complete routing of all the nets.

With the advancement in VLSI fabrication technology, the switching delays of gates have been reduced to the order of picoseconds, and thus interconnection delays are becoming increasingly important in determining the speed of the circuits. In the design of dense high performance circuits, the interconnection delays contribute as much as half of the clock cycle. Thus large interconnection delays can make it impossible to achieve the required clock rate.

In the past, characteristics like connectivity of modules (e.g. macro-cells or standard cells), routability of the nets with the available routing constraints like routing space and pins and pad locations, and layout area were the ones used to perform physical design without any special concern for timing. But this trend has changed. Now, timing is included as one of the most important factors guiding the design process. Performance-driven layout design has received considerable attention in the past several years. Work has been done mostly on the timing-driven placement problem, where a number of methods have been developed for placing blocks or cells in timing critical paths close together. Only limited work has been done for the timing-driven interconnection problem. The objective of our work is to implement a timing-driven routing program for standard cell design methodology that will make use of the timing constraints assigned to each net to keep the maximum delay of any net minimum.

1.2 Standard Cell Design

Standard cell design is the object of this research. In this design style, all cells (flip-flops or gates) have the same height but varying widths. The width of the cells are usually a multiple of some grid unit depending upon the technology. For 2μ MOSIS CMOS technology used in this work, the grid is 8 microns. Cells are placed in an array of horizontal rows, and all interconnections of signal nets are made by channel routing in the spaces between the adjacent rows. For each terminal on one side of a standard cell there is also an electrically common terminal on the opposite side, either of which can be selected for routing. The standard cell layout model is shown in Figure 1.2. The four blocks surrounding the cell rows on the top, bottom, right and left are the external I/O buffers. The rectangular areas in between the cell rows are the routing regions or channels. *Feed-through* cells are inserted within the cell rows in order to provide interchannel routing (or there may be feed-through

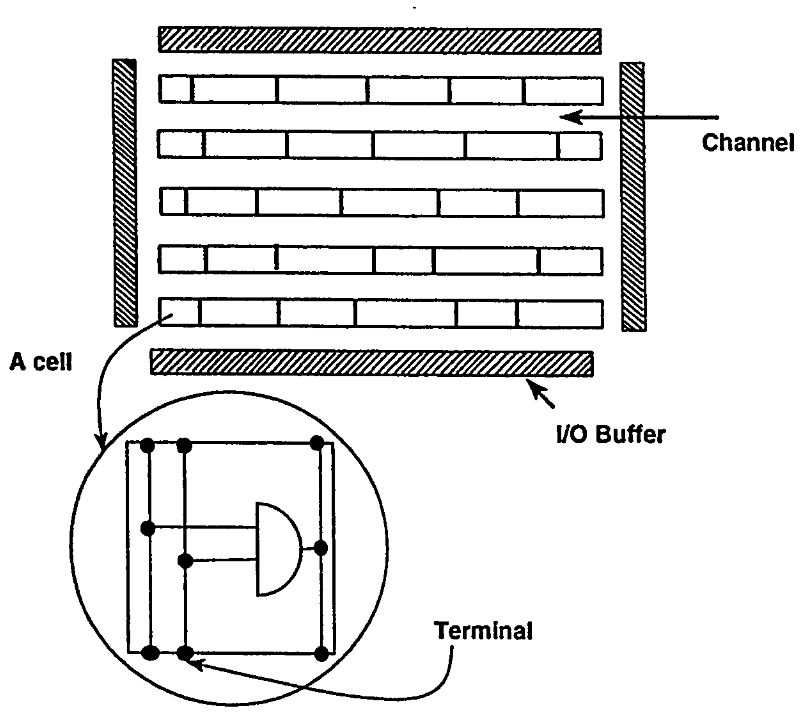


Figure 1.2: Standard cell layout model [2].

pins available within the cells themselves).

The process of routing a standard cell integrated circuit is broken down into two phases. In the first phase, called global routing, nets or net segments are assigned to specific routing channels. As a result of this phase, the interconnection pattern in each channel is defined and is independent of all other channels. In the second phase, called detailed routing, the interconnection pattern in each channel is implemented by assigning the net segments to tracks and columns in the channel.

The interconnection pattern is defined by the set of nets N $= \{n_i\}$ where each net n_i is a set of terminals t_j^i which are to be interconnected. One terminal $s \in n_i$ is a designated source and the remaining terminals are sinks.

Each net can be modeled as a routing graph $G(n_i) = (V_i, E_i)$. Each vertex $v_j \in V$ corresponds to a terminal t_j^i on n_i. A routing solution of a net n_i is a tree in $G(n_i)$ which we call the *routing tree* of the net, connecting all terminals/nodes in n_i. Consider the example in Figure 1.3. Figure 1.3(a) shows the terminals of a single net as the vertical strips on a standard cell design, and Figure 1.3(b) shows the graph modeling this net. The vertices correspond to net segments and edges show the potential connectivity between the net segments. Figure 1.3(c) shows the final net segments based on a routing tree formed out of the routing graph. For the standard cell design style, the channels do not have prefixed capacities. Global routing consists of assigning nets to these channels so as to minimize channel densities and congestion, overall connection length, number of vias and feed-throughs. *Vias*

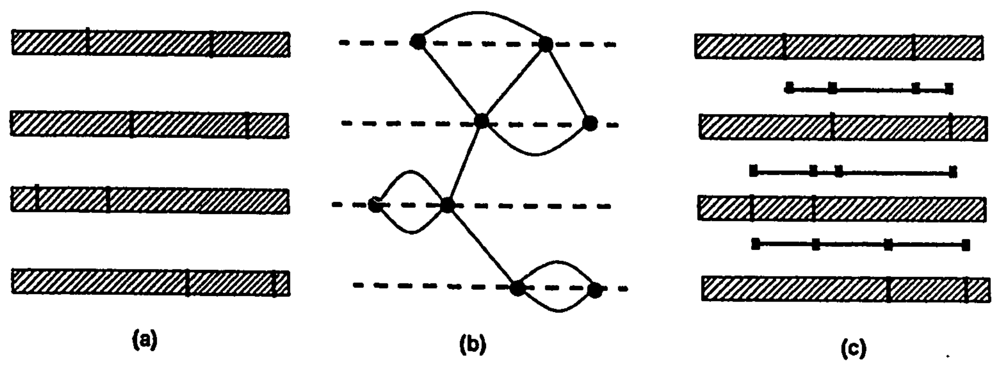


Figure 1.3: A set of net segments [3].

are the holes in the insulating layer that connect conductors of two metal layers generally used for routing. Global routing for standard cell presents two important problems: how many feed-throughs to use and where to place them [6]. Moreover, the more nets there are the more difficult it becomes to assign feed-throughs. For these reasons, a feed-through assignment optimization is needed which eases channel congestion, realizes minimum number of tracks and which does not depend on the routing order. In the same way, a larger via count introduced by a large number of bends in the net segments will result in longer timing delays and longer wire length and decrease in the performance of the circuit.

The objective of this thesis is to develop a timing-driven global routing program that can determine the routes of all nets of a given layout in such a way that a minimum chip area is obtained with minimum timing violation.

1.3 Organization of the Thesis

The thesis is organized as follows: In Chapter 2 we present a thorough literature survey on the global routing problem for standard cell design. It also covers timing-driven approaches used for standard cell design. Chapter 3 discusses the Steiner tree problem which is an essential part of the routing problem. In Chapter 4 we describe an iterative improvement technique called *Tabu Search* which has been used in this thesis to improve the initial global routing solution. We also briefly discuss another iterative technique called *Simulated Annealing.* Chapter 5 describes in detail the global routing technique used in this research. Chapter 6 contains a complete description of the application of Tabu Search technique for improving the initial global routing result. Finally, Chapter 7 provides experimental results, conclusion and future directions for research.

Chapter 2

Literature Review

A very common technique for routing considers the layout as a maze. This technique may be used for both global and detailed routing. Finding a path to connect any two points belonging to the same net is similar to finding a path in the maze. The most widely used algorithm used for maze routing is Lee algorithm [7]. It attempts to find a path between two points on a grid that has obstacles (functional cells or already placed nets). The major characteristic of Lee algorithm is that if a path exists between two points, then it is definitely found. Furthermore, it is guaranteed to be the shortest available path. The running time of the algorithm is high i.e., if L is the length of the path to be found, then the processing time is $O(L^2)$. In addition, the memory requirement for an $N \times N$ grid plane is $O(N^2)$. Some variations of Lee algorithm for speed improvement have been suggested by Hadlock [8] and Soukup [9].

Line Search Algorithms, first proposed by Mikami-Tabuchi [10] and Hightower [11], overcome the drawback of large memory requirements of Lee algorithm and its variations. Line search algorithms perform depth-first search in contrast to the breadth-first approach used by Lee algorithm and its variations. Due to this nature of line search algorithms, they do not guarantee finding the shortest path, and may need several backtrackings. They show completion rates similar to Lee algorithm, but with lesser execution time and memory requirement. The reason is that the entire routing space is not stored as a matrix as in the case of Lee algorithm, but the routing space and paths are presented by a set of line segments.

Global routing approaches fall into four general categories: (1) Sequential global routing approach, (2) mathematical programming approach, (3) stochastic iterative approach, and (4) hierarchical approach. A very detailed treatment of these approaches may be found in a recent book of Sait and Youssef [1].

Sequential global routing is a graph based approach. It is the simplest and most widely used approach. After the routing channels (regions) have been identified and the corresponding routing graph constructed, global routing proceeds as follows. Using the channel connectivity graph, we mark for each net those vertices of the graph in which the net has pins. Hence, routing the net is accomplished by finding a Steiner tree (preferably optimal) covering those marked vertices, where a Steiner tree is a tree of minimum total length, or depending on the context, an approximation to such a minimal tree. Two general approaches are possible in this case: (1)

the order dependent approach and (2) the order independent approach. When the available routing space is updated after the routing of each net, the approach is order dependent, otherwise it is order independent.

In the *mathematical programming* approach, global routing is performed as a 0-1 integer optimization program, where a 0-1 integer variable is assigned to each net and each possible routing tree of that net. The layout surface is modeled as a grid graph, where each grid or routing region is represented by a vertex. The arcs linking the vertices carry some weight which correspond to the capacity of the boundary between two nodes in terms of the number of tracks. We need to identify different possible routing trees for each net. We also associate an equation with each net such that only one of the many possible trees is selected. In spite of its elegance in finding a globally optimum assignment of the nets to routing regions, this technique has some major problems. These problems include the identification of a number of Steiner trees for each net, selection of the trees to guarantee the feasibility of the solution, and a large number of integer constraints.

In the *stochastic iterative* approach, the acceptable assignment of the nets is found by iteratively updating the current solution by ripping up and rerouting the nets. Folowing this approach, a simulated annealing (SA) application to global routing was first reported in 1983 [12]. The SA method is a stochastic sequential improvement method characterized by its ability to escape from local solutions and routing order. The SA method gradually reduces randomness in a system and eventually arrives at the optimal solution. In the first reported application of SA, the authors have formulated the problem as an unconstrained integer program, where all edge capacities are equal to one. Nets with only two terminals and routes with just one bend are considered. The cost function used is a sum of the squares of the loads of all individual routing regions.

Another application of simulated annealing technique to global routing has been adopted in TimberWolf package [13]. TimberWolf package uses two metal layers for routing of standard-cell designs. After the initial placement phase, Timberwolf performs global routing in two stages. In the first stage, assignment of nets to the horizontal channels is done in order to minimize the overall channel densities. All nets that may be assigned to an adjacent channel (switchable nets) are identified in this stage. In the second stage, the aim is to reduce the overall channel densities by changing the channel assignments of the switchable nets. Refinement of the placement takes place after global routing. This is done by randomly interchanging neighboring cells. After each interchange, both stages of the global router are invoked to reroute the nets affected by the interchange. Simulated annealing is used only in the second stage of global routing (as well as the placement refinement phase).

Hierarchical approaches break down the overall global routing problem into subproblems such that the subproblems are solved individually. Solution of the original problem is obtained by combining the solutions of the subproblems.

There are bottom-up and top-down approaches to hierarchical global routing. In the bottom-up approach, grid cells are combined into bigger cells until the entire chip is treated as a super cell. Global routing is performed at each level of the hierarchy for the individual cells considered for grouping. In contrast, for the top-down approach, a hierarchy is formed from super cell to cells, until each cell is an individual grid cell or a small group of individual grid cells. The design floorplan usually assists the top-down approach.

Among the several hierarchical formulation reported in the literature, the first one is due to Burstein and Pelavin [14]. Sadowska [15] and Lauther [16] have also proposed approaches similar to hierarchical global routing, and these were reported to produce better solutions than all other approaches.

Finding a Steiner tree for each net is an important problem in global as well as detailed routing. In one of the works [17] on this problem, a bottom-up hierarchical approach to building Steiner tree has been adopted. Based on a variation of a minimum spanning tree algorithm, a collection of partial steiner trees are formed at each level of the hierarchy. At a higher level of the hierarchy, these lower level Steiner trees are merged such that the duplicate edges are removed and cycles are avoided. Finally the tree corresponding to the top level of the hierarchy is the required Steiner tree for the current net.

A performance oriented rectilinear Steiner tree (POMRST) heuristic has been reported by Andrew Lim et.al. to solve the rectilinear Steiner tree problem [18] which shows favorable results compared with existing techniques. The heuristic works iteratively by growing the Steiner tree one edge at a time. There are two stages during each iteration. In the first stage a spanning tree is constructed which is based on Prim's algorithm. In the second stage, the spanning tree edges are used to direct the selection of Steiner edges one at a time. The paper however does not provide much details about the delay model used to perform timing analysis.

In another approach [19], the idea of minimum spanning tree and Steiner tree algorithms has been used to formulate a routing tree with bounded radius and bounded interconnect delay.

Recently, a neural network technique has been used for global routing [20]. In this paper, the authors have suggested a two-layer neural network. The purpose of the first layer is to minimize net lengths and attain a uniform distribution of the nets over the routing channels. The second layer is meant to carry out effectively the channel capacity constraints.

With the advancement in VLSI technology, the interconnection delay has become significantly important in determining the circuit speed. Therefore, considerable focus has been given to performance-driven layout design during the past several years. Much of the efforts has been made in timing-driven placement problem, but limited progress has been reported for the timing-driven routing problem. The factors that lead to minimizing the interconnection delays are; the overall interconnection length, number of wiring bends, and other electrical characteristics like wire resistance and capacitance. A recent approach [19] has formulated a global routing technique which minimizes the total interconnection length as well as maximum wiring delays. In another approach [21], timing analysis is used in global routing. While routing a connection, the slacks of the affected paths are updated at the corresponding path sinks. The maximized cost function is the slack of the longest path. The timing-driven global routing problem has also been formulated as a multiterminal, multicommodity flow problem with integer flows and additional timing constraints [22].

In the routing problem of physical design, the *channel routing* strategy is popular for its simplicity and efficiency. It is concerned with the actual interconnection of the nets after the initial routing plan has been established by the global router. The channel router takes each routing region and specifies for each net passing through that region particular tracks and wire segments that interconnect the net. Given sufficient routing space, channel router guarantees 100% completion of the routing. Due to this fact, a majority of modern IC (integrated circuit) routing systems adopt channel routing.

Numerous channel routing techniques are based on the left-edge algorithm, which route the channel one track at a time, and for which now exist many variations and extensions in the literature as can be seen in [23] and [24].

When the routing region is such that terminals lie on all the four sides of the region with no internal obstruction, then routing in such a region is called switchbox routing. Switchbox routing is a harder problem than channel routing. An efficient algorithm has been reported by Luk [25] which is basically an extension and modification to the greedy heuristic of Rivest and Fiduccia [26].

Chapter 3

Rectilinear Steiner Tree Problem

3.1 Introduction

The Steiner tree problem has been studied for the past three decades and has received considerable attention due to its involvement in a number of applications. Among these are planning problems in transportation networks, installing communication cables, building systems, printed circuit boards and VLSI design [4].

The Steiner problem comes in two different flavors: Euclidean Steiner problem and rectilinear Steiner problem. Consider two points i and j in the Euclidean plane and let their coordinates be (x_i, y_i) and (x_j, y_j), respectively. We define the cost of the edge connecting the two points as the Euclidean distance $\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ between i and j. Then, if V is a set of points in the plane, the Euclidean Steiner problem is the problem of connecting together the vertices in

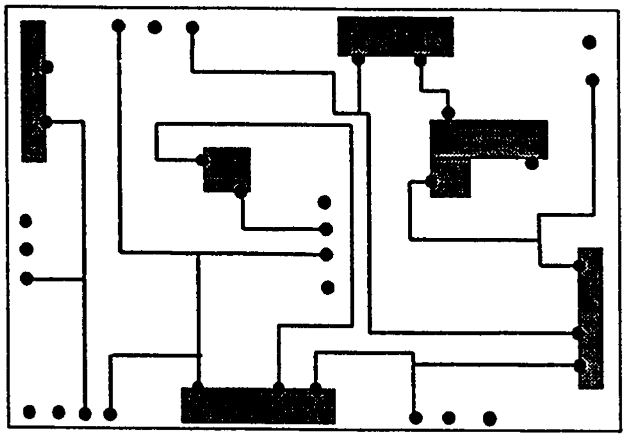


Figure 3.1: A printed circuit board.

$V \cup S$ so as to minimize the total cost of the edges used, where S is the set of Steiner vertices, possibly empty. If $| V | = 2$ then the solution to this problem becomes a shortest-path calculation. Otherwise, the solution is a minimum cost spanning tree (MST) on some set of vertices $V \cup S$.

The rectilinear Steiner problem is the same as the Euclidean Steiner problem except that the cost of the edge connecting any two points i and j together is given by the Manhattan distance $| x_i - x_j | + | y_i - y_j |$. This problem is common in printed circuit board design and in VLSI design. In both, the wire length between components to be connected is minimized. Figure 3.1 shows an example.

Both of the above described problems are NP-hard [27]. They have received a fair amount of attention in the literature and many heuristic algorithms have been proposed.

A comprehensive survey of work relating to the Euclidean and rectilinear Steiner problems has appeared in [4] and [28]. This chapter presents an overview of the rectilinear Steiner problem in general and its relation to the global routing problem in particular.

3.2 Rectilinear Steiner Tree Problem: An Overview

Since a comprehensive survey of work relating to Euclidean and rectilinear Steiner problems has recently been given by Hwang and Richards [4], we shall not give a complete literature survey here but instead concentrate on the application of rectilinear Steiner problem to the global routing problem in VLSI design.

Connecting n points of a net to make them electrically equivalent is a fundamental problem in a VLSI layout. The objective function that is usually considered is the least amount of wire in forming the connection. Due to a number of technological and engineering reasons, the segments are laid out in horizontal and vertical fashion. That is why the rectilinear distance function is used to calculate the length of segments. The tree that is formed is called a rectilinear Steiner tree constructed using the rectilinear metric.

Finding a rectilinear minimum spanning tree (RMST) of the n given points is a possible approximate solution to the problem. Any "classical" minimum spanning tree (MST) algorithm can be used if the problem is considered as an instance of a complete graph with rectilinear distance edge weights. For instance, Prim's algorithm can be implemented in $O(n^2)$ time [29], i.e., quadratic time which is optimal for a complete graph. There are an infinite number of ways in which two points in an RMST can be connected using shortest rectilinear wires. Even if the wires are restricted to make just one bend there can still be two ways to place each wire. Therefore for a complete graph of n points, there are 2^{n-1} possible placements of the $n-1$ wires which may have several segment overlaps. These segment overlaps must therefore be removed by the circuit designer.

Steiner trees usually provide shorter wirelength. A Steiner tree spans the n points but may contain additional points in the plane as vertices to provide additional internal branching. A Steiner tree refers to a tree of minimum total length or, depending on the context, an approximation to such a minimum cost tree. If overlaps are removed from a RMST, it effectively produces an (approximate) rectilinear Steiner tree (RST), though not necessarily minimal. There is no efficient procedure known for computing a minimum RST. Nonetheless, we formally define a minimum rectilinear Steiner tree problem in the plane as follows. Given a set S of n points in the plane, find a set S' of points that do not belong to S, called *Steiner points*, such that the spanning tree over $S \cup S'$ is of minimum total length, with rectilinear distance edges that connect all the points in set S.

The minimum rectilinear Steiner tree (MRST) problem is a fundamental problem in global routing and wire estimation for VLSI circuit layout, where we are interested in connecting terminals of a net together. Several investigations of the

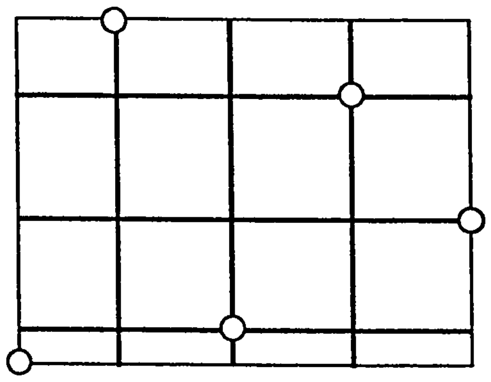


Figure 3.2: Hanan's description of a grid graph [4].

MRST problem have greatly influenced the research progress. Initially, Hanan [30] showed that if one extends horizontal and vertical grid lines through each of the n points in a set S, then there is an MRST whose Steiner points are all chosen from among the intersection points (the *Steiner Candidate set*) in the resulting grid. An example is given in Figure 3.2. Later, Gary and Johnson showed that despite this reduction on the solution space, the MRST problem is NP-complete [27]. This implies that an efficient optimal algorithm for the MRST problem is unlikely to be found. Thus, a number of heuristics have been proposed as surveyed in [28, 31]. In order to attack intractable problems, the basic goal is to formulate *provably good* heuristics, typically in the sense of having bounded worst-case error from optimal. Hwang [32] established that the rectilinear *minimum spanning tree* (MST) over S is a fairly good approximation to the MRST with a worst case performance ratio of 3/2; i.e., if $c(T)$ represents the total cost of a tree T, $c(MST)/c(MRST) \leq 3/2$. Due

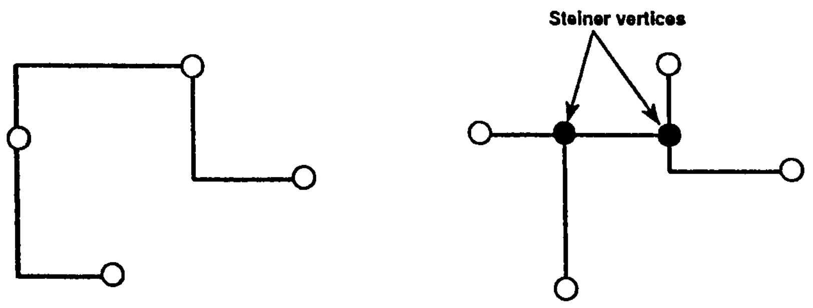


Figure 3.3: MST (*left*) and MRST (*Right*) for the same four points.

to immense time required to obtain the optimal solution for the MRST problem, all heuristics tabulate their performance by comparing themselves against this result. That is, any MST-based strategy which improves upon an initial MST topology (arrangement of points on the grid) and rearranges edges to induce Steiner points, will also have performance ratio of at most 3/2. Thus, a number of Steiner tree heuristics resemble classic MST construction and it remains an open question to find a heuristic method with performance ratio strictly less than 3/2. Figure 3.3 shows an MST and MRST for the same four-pin net.

An MST-based solution method may not be appropriate for VLSI routing applications. It has been shown in [33, 34] that the optimal Steiner tree, as well as heuristic MST-based rectilinear Steiner tree, will have a linear expected number of Steiner points. But having too many Steiner points is undesirable because each Steiner point is equivalent to introducing a via and vias are expensive in terms of delay and performance of the circuit. Thus, based on this fact, a more direct approach to adding Steiner points have been suggested by Kahng and Robbins [35]. Their *iterated 1-Steiner* heuristic repeatedly finds the next best possible Steiner point and adds it to the point set until no further improvement is possible. The performance ratio of this approximation algorithm is never as bad as 3/2 and it is proved to be not greater than 4/3 on the entire class of instances where the cost ratio $C(MST)/C(MRST)$ is 3/2.

The Steiner tree algorithm is the fundamental part of a global routing algorithm in VLSI physical design. In the standard cell (objective of our work) and macro-cell layout styles, global routing is accomplished in a routing graph extracted from a given cell placement [36]. The graph edges correspond to routing channels, while some vertices correspond to intersections of channels and others correspond to interior points in the channels. A usual strategy for global routing consists of two phases [36]. A number of alternative routes are generated for each net in the first phase. The nets are independently treated one at a time with the objective of minimizing the total edge cost of the routing path in the graph. In the second phase, a specific route is selected for each net based on a criteria of channel capacity, overall chip area and/or total interconnection length.

Most existing Steiner tree algorithms of wire-mode or timing-driven global routers take the minimization of total wire length as the objective. Only a few consider the timing constraint during the construction of global routing tree.

3.3 Summary

This chapter provided an overview of the rectilinear Steiner tree problem and its relation to the global routing algorithms in VLSI design. In Chapter 5 we will present a timing-driven global routing scheme based on the technique of dividing a net into zones and thus reducing the complex problem of determining the routing tree of a net, into simpler smaller problems of routing net segments in different zones. An iterative improvement technique has been applied to improve the routing result in terms of delay bounds and channel densities.

Chapter 4

Tabu Search and Simulated Annealing

4.1 Tabu Search

4.1.1 Introduction

Tabu search is a general iterative technique, that was originally proposed by Glover [37, 5, 38] for finding good solutions to combinatorial optimization problems. This simple and elegant technique is regarded as a higher-level method, or meta strategy for solving optimization problems. It has been successfully applied to a number of computationally hard problems like graph coloring [39], graph partitioning [40], VLSI placement [41], circuit partitioning [42], maximum independent set problem

[43], etc.

In general terms, tabu search is an iterative procedure that starts from some initial feasible solution and attempts to determine a better solution in the manner of a greatest-descent algorithm. It does not employ pure randomization like simulated annealing nor does it take the restrained approach that the proper rate of descent will make the final solution closer to a global optimum [40]. Instead of terminating upon reaching a point of local optimality, tabu search structures the operation of its embedded heuristic in a manner that permits it to continue. This is accomplished by forbidding moves with certain attributes (making them tabu), and choosing moves from those remaining to which the embedded heuristic assigns a highest evaluation. In this respect, tabu search employs a mechanism of control which constraints and frees the search process. This corresponds to *tabu restrictions* and *aspiration criteria.* Tabu search uses a flexible attribute-based memory structures to exploit historical search information more thoroughly than using rigid memory structures (such as branch and bound) or by memoryless systems (such as simulated annealing).

4.1.2 Tabu Restrictions

As opposed to the randomness of the search in simulated annealing, tabu search uses a more systematic approach. Instead of choosing a neighboring solution (a solution reachable from the current solution by making a move) at random, tabu search examines a number of neighbors and selects the best neighborhood solution by making a move among all candidate moves. This best candidate solution may not improve the current solution. Selecting the best neighborhood solution is unlike hill-climbing as it might make a down-hill move. Further, each solution in the solution space must be accessible to every other in a finite number of moves. This condition ensures that the neighborhoods cover every solution in such a way that it is possible to find the global optimum, given sufficient time.

With this approach, however, it is possible to reach a local optimum, ascend and then go back to local optimum, thus causing a cycle. Tabu restriction is a device to avoid such cycling by making selected attributes of these moves tabu (forbidden). Tabu list is a list containing forbidden moves. The tabu list is initially empty, constructed in the k first iterations and updated circularly in later iterations. The parameter k is the *tabu list* size. Normally the forbidden moves are the reverse of the k moves last performed. Thus, due to the tabu list, tabu search avoids returning to the solutions just visited, and since degrading moves are allowed, the algorithm has a chance of leaving a local minimum.

The most difficult part of applying tabu search is finding the right size of tabu list. No single rule gives good sizes for all classes of problems. If the tabu list size is too small the search will start cycling, and if it is too large the search might be too restrictive. Therefore, an appropriate list size can be determined by watching the occurrence of cycling when the size is too small and the quality of solution when the size is too large. It is a question of finding the right compromise, typically depending on the specific problem investigated. However, good performance has been achieved with tabu list sizes from 5 to 12. In many applications the tabu list size of 7 seems to be quite effective.

4.1.3 Aspiration Criteria

This component of tabu search introduces diversification and intensification in the search by exploring new regions in the search space. It is a heuristic rule based on cost values, associated with feasible solutions, such that it temporarily overrides the tabu status if the move is sufficiently good. That is, we allow a move currently on the tabu list if it leads to a better solution than the current best solution. The purpose of aspiration criteria is to increase the flexibility of the algorithm while preserving the basic features that allows the algorithm to escape local optima and avoid cyclic behavior. At all times during the search procedure each cost c has an aspiration value that is the current lowest cost of all solutions arrived at from solutions with cost c. The actual aspiration rule is that, if the cost associated with a tabu solution is less than the aspiration value associated with the cost of the current solution, then the tabu status of the tabu solution is temporarily ignored and it is accepted as the next current solution. That is, although the tabu solution is not removed from the tabu list, its tabu status is overridden, and a move to the tabu solution may be made.

It is to be noted that like other optimization procedures, tabu search routine is ignorant of optimal cost values. this results in the procedure being unable to stop. Thus, a stopping criterion must be used, based on a maximum number of moves or a time limit.

4.1.4 Algorithmic Description

A complete description of tabu search is given in [44] which is shown in Figure 4.1. The functions performed by each of its steps are identified. A brief description of these steps is given below:

1. Start with an initial solution which can be generated by a number of ways, randomly or by a constructive process (our initial solution for global routing is generated constructively as explained in Chapter 5). The initial solution is also assumed to be the current best solution.

2. A set of moves is formed. Each move generates a trial solution. Hence we get a set of trial solutions. The number of moves to make depends upon the specific problem under consideration.

3. Select the move which generates the best solution among a set of trial solutions. Call this move a candidate move, that is, it is a candidate to be an admissible move.

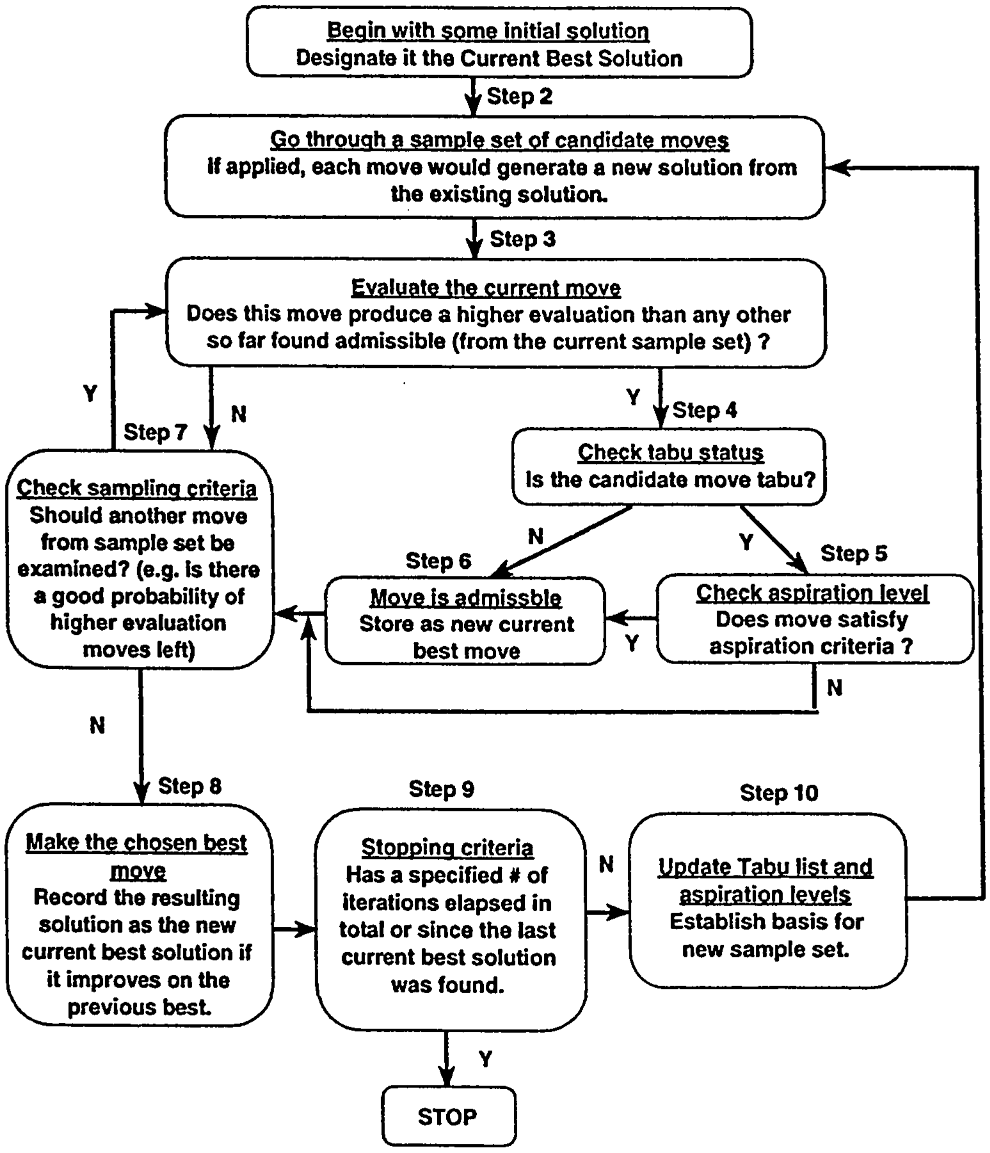

Begin with some initial solution
Designate it the Current Best Solution
Step 2
Go through a sample set of candidate moves
If applied, each move would generate a new solution from the existing solution.
Step 3
Evaluate the current move
Does this move produce a higher evaluation than any other so far found admissible (from the current sample set) ?
Y
Step 4
Check tabu status
Is the candidate move tabu?
N
Y
Step 5
Check aspiration level
Does move satisfy aspiration criteria ?
Y
N
Step 6
Move is admissble
Store as new current best move
Y
N
Step 7
Check sampling criteria
Should another move from sample set be examined? (e.g. Is there a good probability of higher evaluation moves left)
N
Step 8
Make the chosen best move
Record the resulting solution as the new current best solution if it improves on the previous best.
Step 9
Stopping criteria
Has a specified # of iterations elapsed in total or since the last current best solution was found.
N
Y
STOP
Step 10
Update Tabu list and aspiration levels
Establish basis for new sample set.

Figure 4.1: Tabu search algorithm [5].

4. This step includes a key issue of the procedure, which is to establish a basis for deciding if a move being examined should be classified as tabu. It is checked that whether the candidate move is in the tabu list or not. As mentioned in the previous section, the tabu list records selected attributes of each move made. If the move is found to be tabu then go to step 5, otherwise go to step 6.

5. In step 4 if the candidate move is found to be tabu then check the aspiration criteria (as explained in Section 4.1.3). If the candidate move passes the aspiration criteria go to step 6 otherwise go to step 7.

6. The candidate move is considered admissible (i.e., either it is not tabu, step 4, or tabu but passes the aspiration criteria, step 5). Store this move as new current best move and go to step 8.

7. Select next best move from the sample set formed in step 2 or generate a new set of neighbor solutions and select the best among them and go back to step 3, otherwise go to step 8.

8. If the current best solution improves the previous best solution, store it as the best solution found so far.

9. Check for the stopping criterion. If the total number of iterations have been elapsed than stop, otherwise go to step 10.

10. Update tabu list and aspiration levels, that is, include the current best move into the tabu list and the objective function value as its aspiration value for the next iteration. Start new iteration by going back to step 2.

4.2 Simulated Annealing

4.2.1 Introduction

Simulated annealing (SA) was proposed by Kirkpatrick et al. [45] as a heuristic algorithm for various combinatorial optimization problems encountered. It is a stochastic iterative search method characterized by its *hill climbing* feature, i.e., the acceptance of new state of the problem with a non-zero probability which might increase the cost. These moves are controlled by a temperature parameter, in analogy with temperature in the annealing process. At high temperature, the search is almost random, while at low temperature the search becomes alomost greedy. At zero temperature, the search becomes totally greedy, i.e. only good moves are accepted. Thus, the SA method gradually reduces the randomness in the system and arrives at the optimized solution [46]. The SA method is a general heuristic search procedure that has been applied to the travelling salesman problem, the network partitioning problem, large scale combinatorial optimization problems such as placement and routing in VLSI and neural networks [1].

4.2.2 The Algorithm

The Simulated annealing algorithm is shown in Figure 4.2. The SA method is based on the *Metropolis* procedure [47] which simulates the annealing process at a given temperature T (Figure 4.3). The *Metropolis* procedure corresponds to the acceptance functon **accept** in Figure 4.2. This function determines the acceptance (or otherwise) of the new state S_{new} of a given solution S. The prime feature of this function is that, even if the new solution is worse than the previous one, it allows on a probabilistic basis to escape from locally optimized values. The probability of accepting a new state depends on the value of the temperature parameter T. The updating rule for T is given below.

$$T_{new} = \alpha * T; \qquad 0 < \alpha < 1.$$

In the function **accept**, if the cost of the new solution S_{new} is better than the cost of the current solution S, then certainly the new solution is accepted. However, if the new solution has an inferior cost, the parameter T plays a fundamental role. If T is very large, then the random number r (uniformly distributed between 0 and 1) is likely to be less than y and a new state is almost always accepted irrespective of ΔC. If T is small close to 0, than all states with $\Delta C > 0$ have smaller chances of satisfying the test for smaller values of T.

The main parameters of the SA algorithm are:

```
    algorithm Simulated Annealing($S_0, T_0$);
/* $S_0$ is the initial solution and $T_0$ is the initial temperature */
    begin
        $T := T_0$;
        $S := S_0$;
            while("stopping criterion" is not satisfied){
                while("inner loop criterion" is not satisfied){
                    $S_{new}$ = generate($S$);
/* generate is a function which generates randomly a new state $S_{new}$ from state $S$ */
                    if (accept($C(S_{new}), C(S), T$) = 1)
                        $S = S_{new}$;
            }
            $T$ = update($T$);
        }
    end;
```

Figure 4.2: Simulated annealing algorithm.

```
    accept($C(S_{new}, C(Z), T$)
    {
        $\Delta C = C(S_{new}) - C(Z)$;
        $y = min(1, exp\{-\Delta C/T\})$;
        $r = random(0, 1)$; /* returns the number between 0 and 1 */
        if ($r < y$)
                return(1); /* accept the new state */
            else
                return(0); /* reject the new state */
    }
```

Figure 4.3: The accept function.

1. The initial temperature.
2. The new state generation function (**generate**).
3. The acceptance function (**accept**).
4. The temperature update function (**update**).
5. The internal loop exit criterion.
6. The external loop exit criterion.

The implementation details of the above characteristics are discussed in Chapter 6.

4.3 Summary

In this chapter, we have described two general iterative algorithms: tabu search and simulated annealing. Both are powerful optimization technique that are easy to implement, may escape local optima, and may also cope with complex problems involving multiple design criteria and constraints. Furthermore, they may be applied to a diverse variety of combinatorial optimization problems. Both these algorithms have been applied to iteratively improve our initial global routing solution as explained in Chapter 6. A comparative study of their results is also given.

Chapter 5

Global Routing for Standard-Cell Design

5.1 Introduction

Spanning tree based routing has proved useful in cell based design. For a given circuit, there are several possible spanning trees for each net. Many global routing methods are based on constructing a spanning tree for each net [48]. In the Timber-Wolf package [13] minimum weight spanning trees, one for each net, are constructed and then passed through an iterative improvement step using simulated annealing technique. Simulated annealing technique is used to search for a better channel assignmentt of the switchable segments.

Generally all global routers intend to choose net segments in such a way that wiring congestion will be distributed between the channels. Some approaches other than building minimum spanning tree (MST) for the nets use some modification of the basic MST and Steiner tree algorithms to construct a routing tree that simultaneously minimizes both total wire length and maximum interconnect delay [19].

Our goal is to develop a routing scheme that could minimize delays as well as routing area. We shall develop a performance-driven global router so that it can be integrated with another already developed performance-driven placement tool, and can later be connected to the OASIS [49] system. OASIS (Open Architecture Silicon Implementation Software) is a system for standard cell IC design. The tools integrated into the OASIS system have been developed to automatically translate high-level descriptions of integrated circuits into testable physical layouts, using pre-designed standard cells. Therefore, we will take our required input files for cell placement and netlists from the OASIS system. The output of the global router will then be transformed to a format suitable to be used by the detailed router of OASIS to produce the final layout of interconnection nets. The details of all these steps are given inthe following sections.

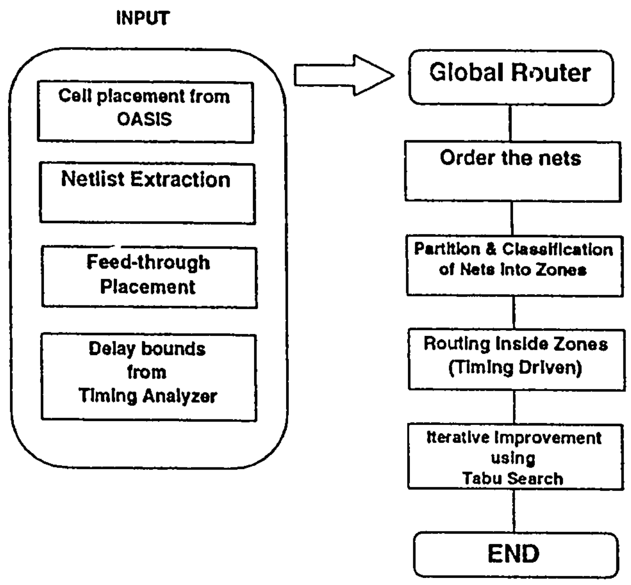

INPUT
Cell placement from OASIS
Netlist Extraction
Feed-through Placement
Delay bounds from Timing Analyzer
Global Router
Order the nets
Partition & Classification of Nets into Zones
Routing Inside Zones (Timing Driven)
Iterative Improvement using Tabu Search
END

Figure 5.1: Main steps of global router.

5.2 Global Routing

The flowchart of our global routing approach is given in Figure 5.1. Each stage is explained in the coming sections.

5.2.1 Determination of Timing bounds on Nets

A signal starting from a source reaches a destination (sink) after traversing one or more circuit modules. A *path* is defined as a sequence of such circuit modules. The start point is an input pad or storage element output pin. The end point is an output pad or storage element input pin. Thus, there can be four different paths

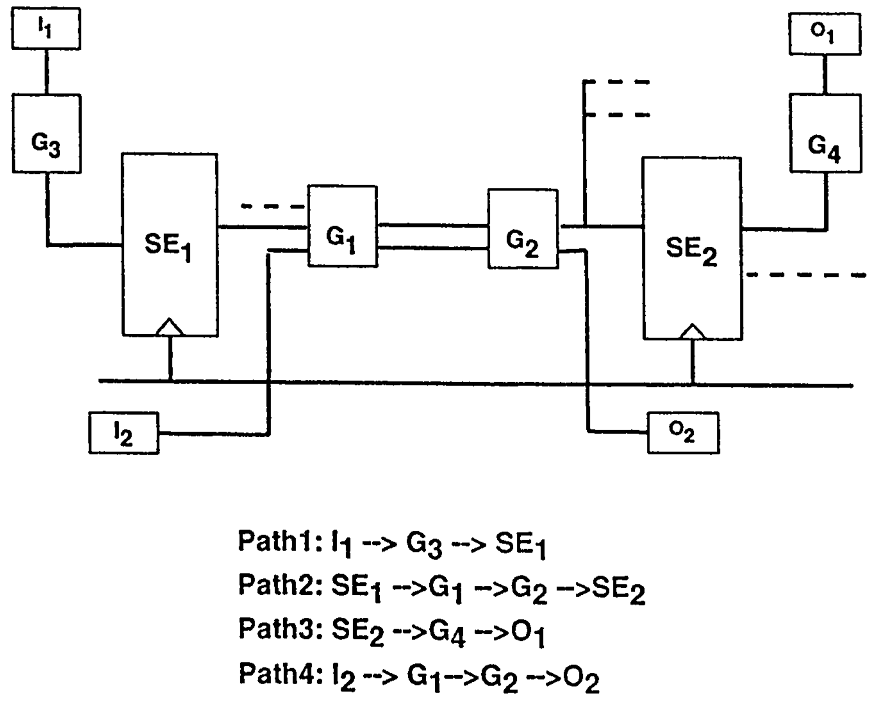

I_1
O_1
G_3
G_4
SE_1
G_1
G_2
SE_2
I_2
O_2
Path1: I_1 --> G_3 --> SE_1
Path2: SE_1 -->G_1 -->G_2 -->SE_2
Path3: SE_2 -->G_4 -->O_1
Path4: I_2 --> G_1-->G_2 -->O_2

Figure 5.2: Four types of paths.

in a VLSI circuit. These paths are shown in Figure 5.2. All storage elements are *flip-flops*.

As the number of paths in a VLSI circuit can be in the millions, it is ususally the case that a very small subset out of all the paths are selected. These are the paths with the largest delays. The path with the maximum delay is regarded as the most critical path. Identification of these critical paths is done by a timing analyzer program. A detailed description of the timing analyzer/predictor is given in [50]. The timing analyzer/predictor program also computes the timing bounds for all the nets in the layout. These bounds are such that if the interconnect delay of each net is within its associated bound, then the circuit will be free from timing problems. The algorithm used to compute net delay bounds is discussed in [51].

5.2.2 Netlist Extraction

The preprocessing stage of our initial global routing starts by first extracting a netlist of the cells, available in the input cell placement file from OASIS [49]. The other inputs to our global router are the standard cell library and the delay bounds of the nets estimated from the timing analyzer program [50]. The netlist creation step involves the scanning of the output of a cell going to the inputs of other cells or to an output pad. At this stage, all pin locations of all cells in a particular row are also determined with respect to the leftmost coordinate (-1,-1). This convention is followed by OASIS for the generation of the layouts. All the required data about a cell are retrieved from the standard cell library containing the relevant cell codes and pin locations in addition to the load factor (LF) and load capacitance (LC) values of each cell. These values are used in delay calculations.

5.2.3 Placement of feed-throughs

The next step after forming the netlist is the placement of the required extra feed-throughs within the cell rows. This feed-through requirement is determined by analyzing the netlist in such a way that the nets requiring feed-throughs are identified and their corresponding requirements are matched against the already available feed-throughs in a particular row. If the available feed-throughs are not sufficient to provide the required routability for the nets, then the required number of extra feed-throughs is determined for each particular row. These extra feed-throughs are then inserted in the cell rows by distributing them on an average distance depending upon the particular cell-row width. At the same time, the pin locations of the cells are also adjusted.

After all the preprocessing steps described above, we move to the timing driven approach to global routing used in our work.

5.3 Timing-Driven Approach to Global Routing

The speed performance of a chip depends to some extent on signal propagation through the interconnects. The smaller feature size has resulted in reduction of switching delays of gates. This has increased relatively the propagation delays through the wires connecting these gates.

Nowadays, it is rare to find a placement program that does not take into consideration timing issues of the circuit. However, routing did not receive similar attention. We believe that timing of the layout can be further improved if timing critical nets are given preferential treatment during routing. Such an approach has been taken in this work. In this section, we shall describe how timing considerations are included during global routing.

5.3.1 Delay Model

The delay model used in this thesis is the *Linear Model.* As per this model, it is assumed that a signal traveling from the source arrives at all the sinks at the same time. The overall delay for any path π, is given by the following equation:

$$T(\pi) = \sum_{c\in\pi} S_c + \sum_{n\in\pi} I_n \tag{5.1}$$

where S_c is the switching delay of cell c and I_n is the interconnect delay of net n. Based on the lumped RC-model, S_c is defined as follows:

$$S_c = BD_c + LD_c \tag{5.2}$$

where BD_c and LD_c are the base delay and load delay, respectively, of cell c. The base delay is obtained as shown below:

$$BD_c = LF_c \times LC_c$$

where the terms LF_c and LC_c denote the load factor of cell c and the total load capacitance at the input pins of cells driven by cell c.

The interconnect delay of net n is computed by the following equation:

$$I_n = LF_c \times C_n + R_n \times C_n + R_n \times LC_c \tag{5.3}$$

where, the terms C_n and R_n represent the capacitance and the resistance respectively of net n.

The interconnect capacitance and resistance for net n are computed as given below:

$$C_n = Area\_capacitance + Fringe\_capacitance \tag{5.4}$$

where,

$$Area\_capacitance = [C_{am1} \times L_{m1} + C_{am2} \times L_{m2}] \times w \tag{5.5}$$

and

$$Fringe\_capacitance = 2 \times [(w + L_{m1}) \times Cf_{m1} + (w + L_{m2}) \times Cf_{m2}] \tag{5.6}$$

$$R_n = \frac{R_{m1} \times L_{m1} + R_{m2} \times L_{m2}}{w} \tag{5.7}$$

where,

C_{am1} = Plate Capacitance/Area of metal m_1,

C_{fm1} = Fringe Capacitance/Length of perimeter of metal m_1,

C_{am2} = Plate Capacitance/Area of metal m_2,

C_{fm2} = Fringe Capacitance/Length of perimeter of metal m_2,

R_{m1} = Sheet Resistance of metal m_1,

R_{m2} = Sheet Resistance of metal m_2,

L_{m1} = Length of metal m_1,

L_{m2} = Length of metal m_2,

w = Width of interconnects.

Next, we illustrate the delay computation for the path π shown in Figure 5.3 through an example. Using Equations 5.2 and 5.3, the switching delay and interconnect delay of the path π of Figure 5.3 can be obtained as follows:

$$\sum_{cell \in \pi} S_c = S_{SEQ_1} + S_{G_1} + S_{G_5} + S_{SEQ_2} \tag{5.8}$$

and

$$\sum_{cell \in \pi} I_n = I_{net_{SEQ_1}} + I_{net_{G_1}} + I_{net_{G_5}} + I_{net_{SEQ_2}} \tag{5.9}$$

The interconnect delay of some net, say net_{G1} in Figure 5.3, is computed by the following equation.

$$I_{net_{G_1}} = LF_{G_1} \times C_{net_{G_1}} + C_{net_{G_1}} \times R_{net_{G_1}} + LC_{G_1} \times R_{net_{G_1}} \tag{5.10}$$

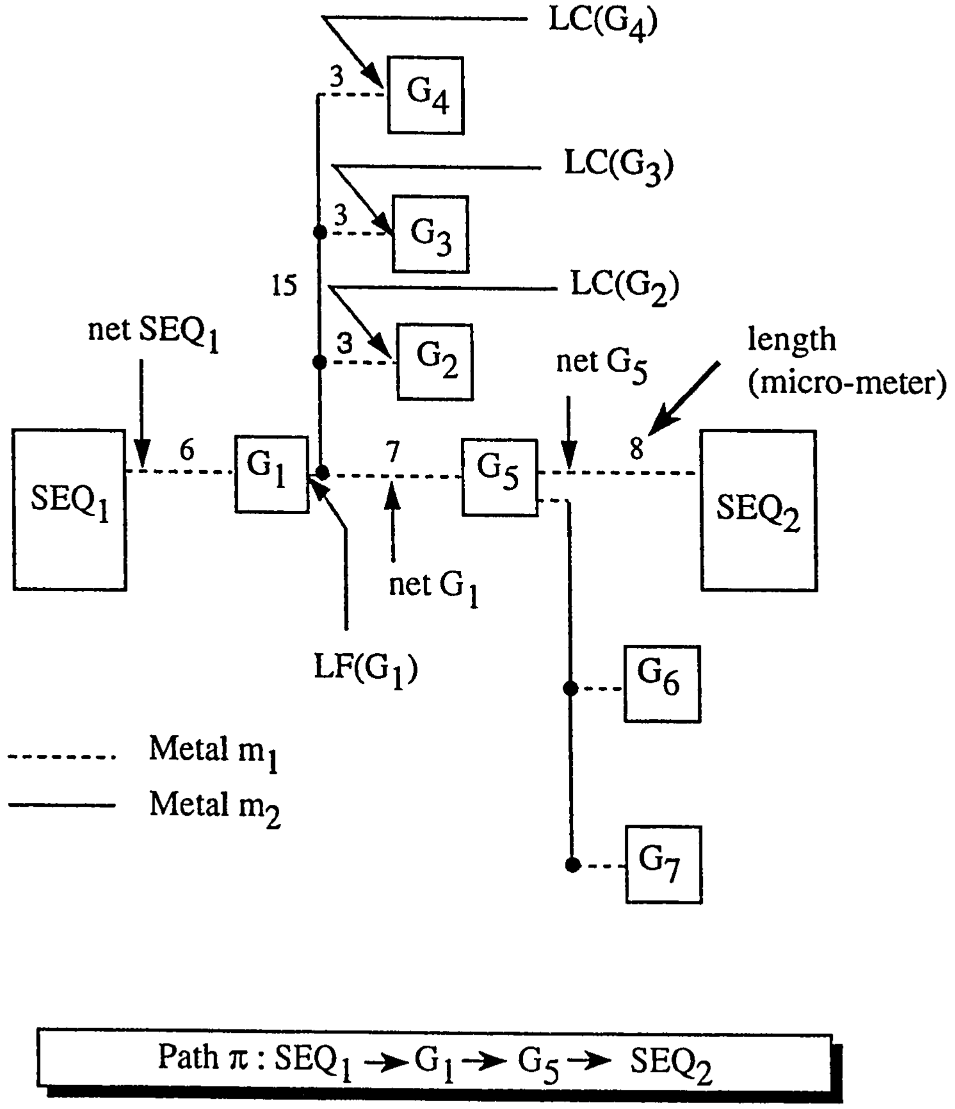


Figure 5.3: Delay model used in this work.

where, the load capacitance of cell G_1 is obtained as follows:

$$LC_{G1} = LC_{G_2} + LC_{G_3} + LC_{G_4} + LC_{G_5} \tag{5.11}$$

Metal_Type	Area_Capacitance $(10^{-4}pF/\mu^2)$			Fringe_Capacitance $(10^{-4}pF/\mu)$		
Range	Min	Typ	Max	Min	Typ	Max
Metal1	0.21	0.23	0.26	0.75	0.79	0.82
Metal2	0.13	0.14	0.15	0.78	0.81	0.85

Table 5.1: Area and fringe capacitance values.

	Sheet_Resistance $(ohms/square)$		
Range	Min	Typ	Max
Metal1	0.050	0.055	0.060
Metal2	0.022	0.028	0.033

Table 5.2: Sheet resistance values.

Different values for capacitance and sheet resistance are given in Tables 5.1 and 5.2 respectively. The width of interconnects for the circuit technology used is w = 3μ. After combining the constant terms in Equations 5.5, 5.6 and 5.7, these equations may be re-written as follows:

$$Area\_capacitance = 0.000078 \times L_{m1} + 0.000045 \times L_{m2} \tag{5.12}$$

$$Fringe\_capacitance = 0.000164 \times L_{m1} + 0.000170 \times L_{m2} + 0.001002 \tag{5.13}$$

$$R_n = 0.020 \times L_{m1} + 0.011 \times L_{m2} \tag{5.14}$$

After combining all the constant terms, the new equations can be written as follows:

$$C_n = CONST_1 \times L_{m_1} + CONST_2 \times L_{m_2} + CONST_3 \tag{5.15}$$

and,

$$R_n = CONST_4 \times L_{m_1} + CONST_5 \times L_{m_2} \tag{5.16}$$

where L_{m_1} and L_{m_2} are the lengths of metals m_1 and m_2 respectively. The new constant terms are,

$\mathrm{CONST}_1 = 0.000242\mathrm{pF}/\mu$

$\mathrm{CONST}_2 = 0.000215\mathrm{pF}/\mu$

$\mathrm{CONST}_3 = 0.001002\mathrm{pF}$

$\mathrm{CONST}_4 = 0.020\Omega/\mu$

$\mathrm{CONST}_5 = 0.011\Omega/\mu$

From Figure 5.3, L_{m_1}=16μ and L_{m_2}=15μ. Let the values of load capacitance for the cells G_2, G_3, G_4, and G_5 be as given below:

$\mathrm{LC}_{G_2} = 0.068\mathrm{pF}$,

$\mathrm{LC}_{G_3} = 0.085\mathrm{pF}$,

$\mathrm{LC}_{G_4} = 0.091\mathrm{pF}$,

$\mathrm{LC}_{G_5} = 0.087\mathrm{pF}$,

and the load factor be,

$\mathrm{LF}_{G_1} = 4.92\mathrm{ns/pF}$, Then the load capacitance of the driving cell G_1 will be $LC_{G_1} = 0.331$pF. The interconnect capacitance and resistance are found to be $C_{net_{G_1}} = 0.008099$pF, and $R_{net_{G_1}} = 0.485\Omega$ respectively. By substituting these values in Equation 5.10, the interconnect delay through net_{G_1} is obtained as follows.

$$I_{net_{G_1}} = 4.92 \times 0.008099 + 0.008099 \times 0.485 + 0.485 \times 0.331$$

which is equal to 0.20431 ns.

Based on the above calculations and experimental results obtained from actual VLSI circuits, it was observed that the load delay component contributes the most to the interconnect delay. The contribution due to interconnect resistance was found to be an order of magnitude smaller than that due to load factor. Before proceeding further, we show that it is safe to ignore the effect of interconnect resistance in the delay computation by the following example.

Let us assume a chip of size $4000 \times 4000\ \mu m^2$ and the interconnect length for a 2-pin net to be $L_{m1} = 2000\mu$ and $L_{m2} = 2000\mu$ for horizontal and vertical connections respectively. Then, the interconnect resistance will be:

$$R_n = 0.02 \times 2000 + 0.011 \times 2000 = 62\Omega$$

Now, let us compute the various components of interconnect delay assuming $LC_c = 0.051pF$, $LF_c = 4.92K\Omega$ and $C_n = 0.012pF$.

$$LF_c \times C_n = 4.92 \times 0.012 = 0.05904ns$$

$$R_n \times LC_c = 62 \times 0.051 = 0.003162ns$$

$$R_n \times C_n = 62 \times 0.012 = 0.000744ns$$

It is evident from the above computation that the delay due to interconnect resistance is very negligible and can be ignored.

5.3.2 Objective Function of the Initial Global Routing

In order to judge the quality of the solution of an optimization problem, it is of utmost importance to have some criteria that will reflect the quality of the solution obtained. Since global routing is an optimization problem with multiple objectives, the choice of proper cost function(s) is not straightforward. This section discusses some of the common objective functions used as a measure to estimate the cost of a global routing solution.

The aim of an objective function is to identify a superior solution. This however, is an optimization problem which is more difficult to solve. Some of the possible criteria for evaluating the quality of a global routing solution can be as follows:

(1) minimize routing area,

(2) minimize total wirelength,

(3) even channel densities,

(4) minimize vias and feed-throughs,

(5) minimize delay bound violations, or

(6) a combination of two or more of the above criteria.

The objectives described above may be conflicting, since we may find that shortest multiterminal length may not always yield the smallest net delay. This is because delays at the sink pins of a multiterminal net depend on how the interconnection tree (topology) is constructed. Similarly, doglegging, introduced in order to minimize the number of horizontal tracks and avoid vertical constraints, increases via count, thus increasing the delay. Feed-through cell insertion in between other cells also introduces some incremental delay as well as area increase as it causes the cell rows to stretch. Even when there is sufficient area for feed-throughs, unrestricted layout of feed-throughs may exert a bad influence on net segment assignment optimization and/or on detailed routing. Hence, we need to formulate a performance-driven routing plan so as to have a balance among the objectives described above.

Since our goal is to perform timing-driven routing, our prime objective is to minimize net delays alongwith the minimization of overall chip area. In the initial global routing stage, we sort the nets based on their criticality as mentioned in Section 5.3.3. Then the nets are routed one at a time based on this sorted order. In this way, the critical nets will have good opportunity to utilize the available feed-throughs. This will result in better routing trees for the critical nets as well as less number of vias. This strategy has resulted in good solutions in terms of timing as shown in Chapter 6. Furthermore, during the routing of each net, our objective is to identify a minimal delay tree.

Our objective in the iterative improvement stage is to improve the overall routing area. This is achieved by the optimal assignment of switchable segments to the channels so as to reduce overall channel density of the layout. The implementation details of the iterative improvement stage are given in Chapter 6.

In the following subsections, we shall provide implementation details of the initial timing driven global routing step.

5.3.3 Ordering of Nets

The nets are routed one at a time, sequentially, based on their criticality. The net criticality is determined by sorting the nets in the ascending order of their delay bounds. Thus the most critical net is routed first. then the next most critical and so on. This routing order is followed based on the fact that if the critical nets are not satisfactorily routed, we are not going to achieve the appropriate routing of the nets which can satisfy the timing requirements.

After the above described preprocessing steps, we move to the routing stage as discussed in the upcoming sections.

5.3.4 Partitioning and Classification of nets

As discussed in Chapter 3, the process of building a Steiner tree for a set of points to be connected is NP-hard. Therefore, to solve such a complex problem, we use a partitioning based approach, which is similar to the one reported by Sugai and Hirata [2]. The reason behind partitioning and classification of nets is to break the complex problem of routing a net into smaller manageable ones. That is, breaking up a net into different segments and zones helps in solving some parts of the routing problem using optimal methods while other parts using sub-optimal methods (as explained in the next sections).

First, all nets are selected one at a time and each net is divided into 2-terminal net segments. For that purpose, we sort the terminals of each net based on their x-coordinates. Next, we classify 2-terminal net segments as follows (see Figure 5.4):

1. One-channel net segments (Figure 5.4 (a)).

   Here the terminals are in neighboring rows and the connecting segment can be laid out on the first attempt in the channel sandwiched by the two cell rows.

2. Two-channel net segments (Figure 5.4 (b)).

   Here the terminals exist in the same row and therefore the corresponding segment is "Switchable". A switchable net segment can be routed at random in either of the channel above or the channel below the cell row. Switchable segments are the object of our iterative improvement step. So, in this way all

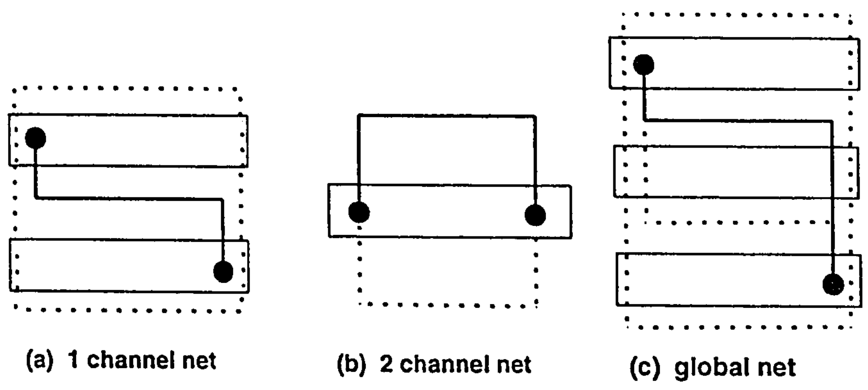


Figure 5.4: A classification of nets.

the switchable segments can be readily identified.

3. Global net segments (Figure 5.4 (c)).

   These are other than the previous two, i.e., the two terminals are separated by one or more intermediate rows. One or more line of feed-throughs are generally needed to route the global net segments.

The rectangular region surrounding each of the above mentioned segments are called 1-channel zones, 2-channel zones, and global zones respectively. Routing is then carried out inside each zone (Figure 5.4) and becomes a straightforward task. Though each two terminal zone can be routed optimally, the overall tree that will result will obviously be far from optimal. Therefore, to keep a routing tree close to optimal, we merge some of the two terminal global zones which are close to each other within some threshold value. This is explained in the next section.

5.3.5 Formation of Feed-through Zones

The partitioning of a net into two-terminal zones simplifies the problem to an extreme. However, this will lead to a poor quality solution (measured in terms of required feed-throughs and overall net length). In order to reduce the number of required feed-throughs, one has to merge some of the global zones into one larger zone of more than two terminals. This is achieved as follows [2].

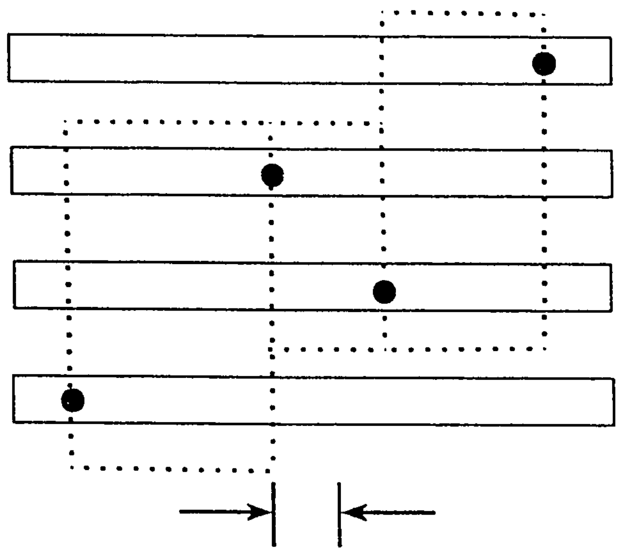
While classifying a net into zones, some of the 2-terminal global zones may be combined to form a zone with multiple terminals. This is called a *feed-through zone.* This feed-through zone helps in reducing the number of feed-throughs that may be required for subnets in neighboring 2-terminal global zones that are quite close to each other. The sampling of a feed-through zone is done by determining if another global zone exists in the same net and the distance between them is within a threshold distance of F_t. This F_t has been taken as the maximum separation of two terminals inside all global zones in a net. After determining the F_t value in the global zones of a net, we proceed as follows:

(a) Expand the present zone to include the terminals of the nearby zone.

(b) Destroy duplicate zones.

(c) Repeat (a) and (b) until all terminals have been processed.

Depending on the value of F_t, feed-through zones change as shown in Figure 5.5. As F_t gets larger, an increasing number of global zones are merged into multi-terminal



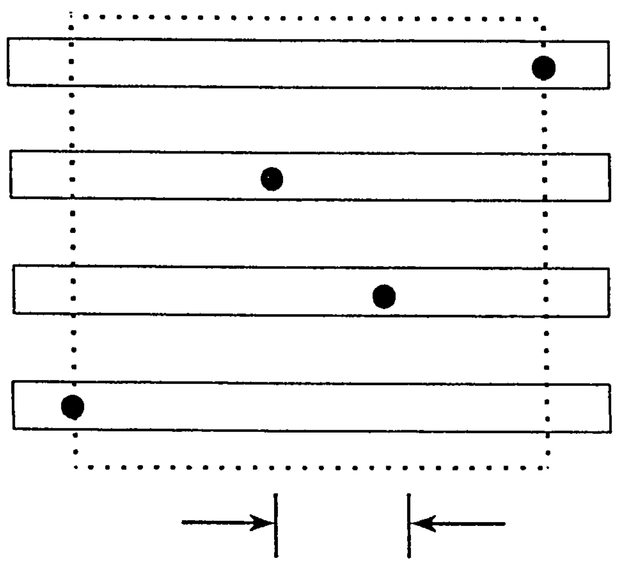


Figure 5.5: Dependence of the size of feed-through zone on feed-through interval.

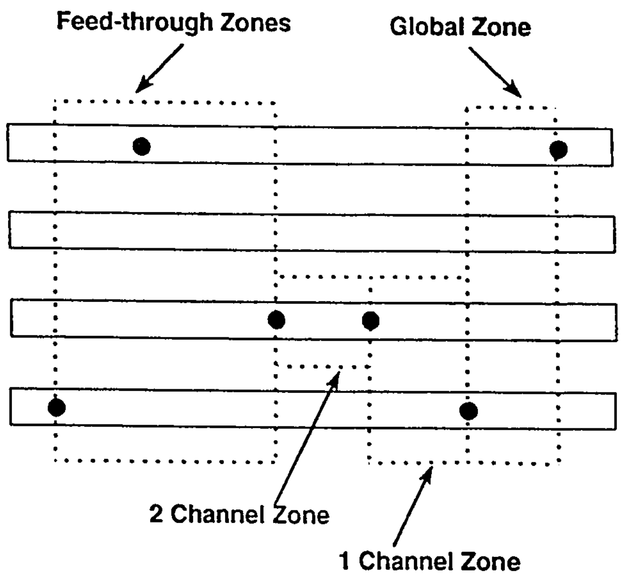


Figure 5.6: Partition of a net into zones.

feed-through zones.

After a net is completely partitioned into different zones, it takes the form as shown in Figure 5.6.

5.3.6 Routing Inside Zones

Routing inside a 1-channel zone is straightforward. The resulting segment is sandwiched between the two cell rows. A 2-channel net or switchable net can also be assigned randomly to any of the channel above or below the row. For that purpose, we generate a random number between 0 and 1. If the generated random number is less than 0.5, the segment is assigned to the channel above the cell row, otherwise it is assigned to the channel below the cell row. The switchable segments are the object of our iterative improvement stage. It has been reported in the literature and has been observed during our experimentation that switchable segments comprise more than 50% of the total number of segments in the layout. Therefore, they are a strong candidate to be optimized. The length of the segments are also recorded as they are laid out. The rectilinear distance function is used to determine the length of the segments[1].

Timing Driven Routing inside global zones

Routing inside a 2-terminal global zone reduces to finding the minimum delay path between the two terminals of the zone. This can be optimally solved using Dijkstra's shortest path algorithm. Dijkstra's algorithm is outlined in Figure 5.7.

In order to apply the Dijkstra's algorithm, we first build a routing grid graph as follows: We extend horizontal lines through the estimated centre inside each channel covered under the 2-terminal bounding box. This central position inside each channel can be taken from the previous estimates of designs of similar size and characteristics. Then we extend vertical lines through each point of the net and through each feed-through pin available inside the net bounding box. The intersection points ($\times$) are the *Steiner candidate points* in the resulting grid (Figure 5.8). It is to be noted that if there are no available feed-throughs inside the terminals' bounding box, then we have to expand our bounding box. This is achieved by

[1]Rectilinear distance between two points p and q is defined as $dist(p, q) = | p_x - q_x | + | p_y - q_y |$.

```
algorithm Dijkstra;
begin
    $S := 0; \overline{S} := N$;
    $d(i) := \infty$ for each node $i \in N$;
    $d(s) := 0$ and pred$(s) := 0$;
    while $| S | < n$ do /* $n$ is the number of nodes in $N$ */
    begin
        let $i \in \overline{S}$ be a node for which $d(i) = \min \{d(j) : j \in \overline{S}\}$;
        $S := S \cup \{i\}$;
        $\overline{S} := \overline{S} - \{i\}$;
        for each $(i,j) \in A(i)$ do
/* $A(i)$ represents the arc adjacency list of node $i$ */
                if $d(i) > d(i) + c_{ij}$ then $d(j) := d(i) + c_{ij}$ and pred$(j) := i$;
    end;
end;
```

Figure 5.7: Dijkstra's algorithm.

extending to the right and left of the bounding box to find available feed-throughs. If available feed-throughs inside the cell row are exhausted, the program terminates with a message to insert more feed-throughs in the feed-through preprocessing stage.

After constructing this grid graph, we determine the shortest delay path between the two terminals using the available Steiner candidate points. During routing, we also keep track of the length of the segments in the horizontal and vertical directions (for metal1 and metal2 respectively) as well as the number of vias, since they will be used in the delay calculation (explained in section 5.3.1). We add a penalty value to the length of the segments whenever they take a turn. This is because each bend introduces a via in the layout and should be regarded as causing some extra delay.

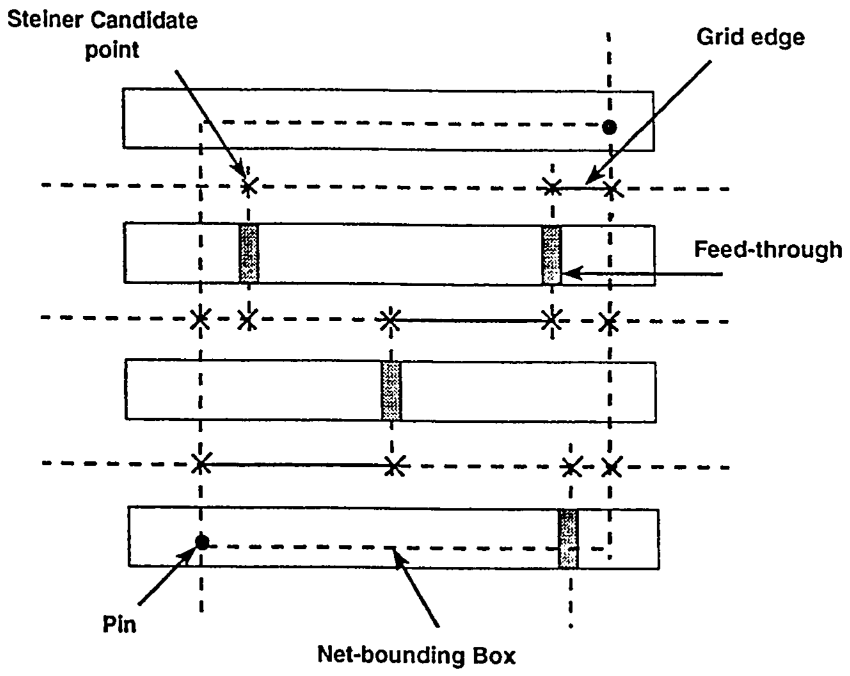


Figure 5.8: Routing inside 2-terminal global Zone.

Therefore, the corresponding delay produced by a via is directly transformed into length values.

Timing Driven Routing inside feed-through zones

A feed-through zone is a multi-terminal zone. The grid graph as shown in Figure 5.9 is formed in the same fashion as explained in the previous section. At this routing stage, we try to determine an MRST (minimum rectilinear Steiner tree) whose Steiner points are chosen from the Steiner candidate set in the resulting grid.

In a first step, we sort all the terminals available in the feed-through zone based on their x-coordinates. Next, we determine a shortest path between the first two terminals of the zone. All the points which lie on this shortest path are treated as potential target points for the routes of the remaining terminals. The remaining terminals of the zone are picked one at a time and shortest paths are determined from the terminal to the potential target points. This process continues until all the terminals are included in the tree. Now this tree may not be a tree of minimal delay. An iterative improvement process is applied to improve delay of this routing tree. This is achieved by ripping up a segment of the current tree and re-routing it in such a way that we get a tree of smaller total delay. In this improvement stage, we rip-up a branch starting from a terminal and try to re-route the terminal through a better path with smaller delay using the same Dijkstra's algorithm. The branch to

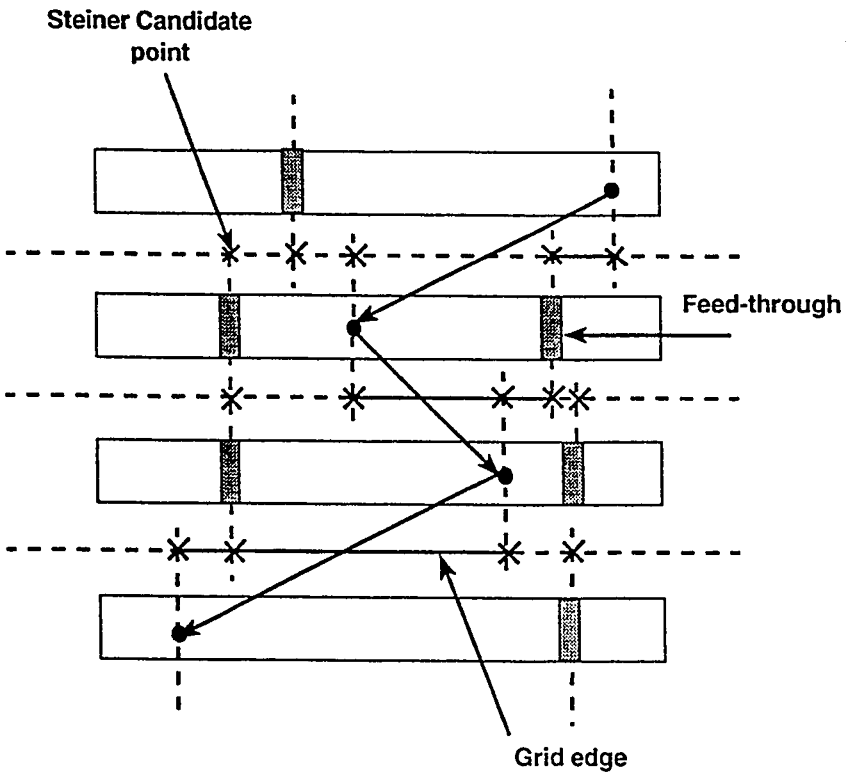


Figure 5.9: Routing inside feed-through zone.

rip-up is selected as follows: we start traversing the tree from a terminal and follow the path of the tree until we hit a point whose fanout is greater than 2. This means that the point is further connected to some other points beside its predecessor and successor. Thus, when a terminal is disconnected from the rest of the tree, we try to find a better path to re-connect the terminal by considering the rest of the nodes of the tree as potential target points. During the improvement stage, we keep the cost (delay) of the initial tree and if some new branch improves on cost, the cost of the tree is updated. The whole process of rip-up and re-route is repeated for $2n$ times, where n is the number of terminals in the feed-through zone.

5.4 Complexity of the Algorithm

The complexity of the algorithm can be analyzed as follows:

The process of partitioning nets into zones is linear in the number of nets. The routing inside a global zone is quadratic since a shortest path algorithm is being used which has complexity of $O(n^2)$, where n is the number of points in the graph. Routing inside feed-through zone is achieved by repetitive application of the shortest path algorithm equal to the number of terminals that are inside the feed-through zone. So the complexity of the algorithm is $O(n^2)$. The memory requirement is linearly related to the number of cells in the circuit.

5.5 Summary

In this chapter, we have presented the global routing scheme that has been used in this research. The chapter also describes various objective functions used in our global routing solution. The delay model used in our performance driven global routing approach is also outlined. In the following chapter, we describe the iterative improvement phase of our global router.

Chapter 6

Channel Density Optimization

6.1 Introduction

In this chapter we describe our approach to improve the initial global routing solution obtained from the first stage, by optimizing the assignment of switchable segments to the channels. Switchable segments account for about 40-50% of the total number of segments as has been reported in the literature and confirmed in this research. Therefore, they are very good candidates for the optimization process. The main concern in this improvement step is to search for a better assignment of the switchable segments in order to reduce the overall channel density as well as to reduce channel congestion. The reduction of channel densities will reduce the routing area required, whereas reducing channel congestion will ease the detailed routing step.

6.2 Optimization of channel density using Tabu Search

In this section we will discuss various issues regarding the application of Tabu search (TS) on switchable segments. The main tasks in TS application are as follows:

- Defining a cost function.
- Starting with a feasible initial solution.
- Defining a neighborhood for a given solution.
- Generation of moves.
- Formulation and maintenance of tabu list.
- Defining an aspiration level criterion.
- Finding a good tabu list size.
- An efficient way to accept moves.

The rest of the chapter will discuss these issues in detail.

6.3 Cost Function

In order to formulate the assignment of switchable segments as an optimization problem, a suitable cost function is required. The optimization technique will then attempt to optimize the value of this function. The cost function C we seek to optimize is the following

$$C = \sum_{i=1}^{n} x_i \tag{6.1}$$

where,

x_i channel density of channel i.

n number of channels.

The cost function represents the summation of the densities of the channels. The *channel density* of a channel can be defined as the maximum number of segments that are crossing a particular virtual grid line on the layout in that particular channel. Thus the summation of channel densities gives us the initial value of the cost function. The overall area of the layout can be reduced, therefore, by reducing this cost value as much as possible.

When a switchable segment is moved from some channel j to some other channel $j+1$, for example, any of the following situations may occur.

1. the channel densities in both channels j and $j+1$ remain unchanged.

2. the channel density remains unchanged in channel j but increases by 1 in channel $j+1$.

3. the channel density in channel j decreases by 1 but remains the same in channel $j+1$.

4. the channel density decreases by 1 in channel j and increases by 1 in channel $j+1$.

Thus, the new cost C_{new} of the solution either increases by 1, decreases by 1, or remains the same.

It follows that the new cost C', after the segment switch, will be

$$C' = C + (x'_j - x_j) + (x'_{j+1} - x_{j+1}) \tag{6.2}$$

where,

x_j	channel density of channel j before segment switch.
x'_j	channel density of channel j after segment switch.
x_{j+1}	channel density of channel j before segment switch.
x'_{j+1}	channel density of channel j+1 after segment switch.

When the cost remains the same, it means that the net segment switch has no effect on the channel density and thus area. We introduce another cost function at this point. This cost function is meant for determining the congestion in a channel between the points defining the span of the net segment. The cost function is computed by taking the difference between the overall channel density of a particular channel and the density of the channel between the points defining the span of the net segment. We first evaluate the cost function for the segment span in the original channel and then for the new channel in which the segment will be switched. The difference in cost (Δc) is determined by subtracting the second cost function value from the first. A negative value of (Δc) indicates that switching the net segment to the new channel places the segment in a channel of less congestion.

6.4 Initial, Current and Best Solutions

Although in theory, the initial solution can be any feasible solution, TS may perform poorly if given an inferior initial feasible solution. The initial solution in our case is made constructively as explained in Chapter 5 but with switchable segments assigned randomly. The cost of this solution is determined as the initial cost. As TS proceeds, we keep two solutions – one is the current solution and the other is the best solution found so far. At the start of TS, best solution is the initial solution and the best cost is the initial cost. The best solution found in a number of iterations is the output of TS, where the number of iterations is a user specified parameter.

6.5 Generation of Moves

The efficiency of Tabu Search strongly depends on the definition of the neighborhood $N(s)$ of a feasible solution, s, and of the Tabu list(s), on their sizes and on the parameter *nbmax* (maximum number of iterations between improvements).

We define for each feasible solution a neighborhood $N(s)$; it consists of the solutions to which one can move in one step of the iterative procedure. The basic step consists in moving from a solution s to another solution s' where s' is the best solution in the neighborhood $N(s)$ (even if it is not as good as s).

For our case, we can define the neighborhood of a solution s as the set of all solutions s' that can be obtained from s by changing the channel of a switchable net segment selected at random from the available list of switchable segments. The cost is calculated after the relocation of the segment. We generate all the candidate moves for iteration *itr* (say) and the best among these is selected, which may not be better than the current solution in terms of cost. The candidate neighborhood list size in our case is set equal to the number of channels in the layout. Moreover, we record in the tabu list the number of the segment which has been relocated.

6.6 Tabu List

As mentioned before, the formulation of tabu list is one of the major decisions in TS application to a problem. In the present implementation we used a single tabu list. This list contains the numbers of the segments which have been selected randomly for the last T iterations. The tabu list size T is an important parameter in TS. If the size is too small the search will start cycling and if it is too large, the search will be too restrictive. Tabu search was applied with different tabu list sizes, such as 7, 12, 15 and 20, on all the benchmarks used. The *magic number* 7 is used for T and it has produced good results for all cases. In this circular tabu list, the addition of another segment number removes the one recorded in its position T relocations ago. If a candidate move is found tabu (segment number present in the tabu list), its aspiration criteria is checked as described next.

6.7 Aspiration Level Criteria

In theory, aspiration level (AL) values occur for each cost ($C(s)$) value encountered during the whole iterative process. A move from s to s' with a tabu status is accepted if $C(s') <= AL(C(s))$. Initially, all ALs are set to a level higher then any possible cost. As moves are made, the appropriate ALs are updated, if necessary. For instance, when the first move is made from a solution of cost 8 ($C(s) = 8$) to a solution of cost 6 ($C(s') = 6$), the AL associated with the cost value of 8 is updated to 6; i.e. $AL(C(s)) = 6$. Since the aspiration level is updated on each acceptance of a move, a tabu move has to do better than the aspiration level recorded when that move was made tabu.

If a tabu solution fails to satisfy the aspiration criteria, the next best candidate solution can be selected from the candidate list of solutions and the process is repeated. Otherwise, another set of candidate solutions may be generated. The number of regenerations can be performed for maximum three times, after which the number of trial solutions are increased by the same amount (the number of channels). The increment to the number of the trial solutions are made for maximum three times.

The above described strategy has proved very useful. Very rare cases have been seen where the number of the trial solutions needs to be incremented three times in order to get an acceptable solution (not tabu or tabu but satisfies aspiration criterion).

The aspiration criteria that are used in our case are described as follows:

- If the tabu solution increases the channel density of the new channel to which the segment is being moved, than do not accept the tabu solution. Otherwise, if the cost of the Tabu solution improves on the cost of the overall best solution, then accept the tabu move.

- If the channel densities in the original and the new channel do not change, then the second cost function is used to determine the relative congestion inside both the channels. If the move to the new channel is a move to a channel of less congestion, then the tabu status of the segment move is ignored and the move is accepted.

6.8 Optimization of channel density using Simulated Annealing

We also experimented with simulated Annealing technique to search for a better assignment of the switchable segments. The following describes our implementation of the various characteristics of the simulated annealing method.

1. **The initial temperature:** The SA method needs to start from a high temperature [46], but if it is too high, it causes a waste of processing time. The temperature value should be such that it allows virtually all proposed new states of the solution in the beginning as the temperature is high. The way the initial temperature parameter is set in our case is as follows.

   Basically the idea is to use the Metropolis function ($e^{\Delta C/T}$) to determine the initial value of the temperature parameter. Initially, before the start of actual SA procedure, we make a fixed number of moves, say M, in the neighborhood of the current solution and determine the respective cost for each of these moves. M is a function of the number of switchable segments. Next we determine the difference of costs (ΔC) for each two successive solutions where,

$$\Delta C_i = C_i - C_{i-1}$$

   The average $\overline{\Delta C}$ is then given by

$$\overline{\Delta C} = \frac{1}{M}\sum_{i=1}^{M} \Delta C_i$$

Since we wish to keep the probability, say P_0, of accepting uphill moves high in the initial stage of SA, we therefore get the value of the temperature parameter by subsituting the value of P_0 in the following expression derived from the Metropolis function.

$$T_0 = \frac{\Delta C}{ln P_0}$$

where $P_0 \approx 1$ ($P_0 = 0.999$).

2. **The new state generation function:** In the case of 2-channel zones (switchable zones), one zone is chosen among them at random. A new state is generated by moving the switchable segment in the channel on the opposite side of the cell row containing the terminals.

3. **Cost Function:** The cost function used is the same used in the Tabu Search implementation i.e. the estimation of wiring area which is approximated by the total channel density, that is, the sum over all channels of the channel density.

4. **The acceptance function:** The Metropolis function is used in its standard form.

5. **The temperature update function:** The parameter α for updating the temperature is a user specified parameter. In the current implementation, α takes on the value of 0.9. At each updated value of the temperature, a number

of state transitions equal to the internal loop count (explained next) are made. These transitions are performed at each stage (or value of T) of the annealing process so as to reach the probabilistic steady state.

6. **The inner loop criterion:** In order to reach the probabilistic steady state at a certain temperature, the number of state transitions needs to be on the average equal to the number of transitions needed for each element to pass through all its possible states [52]. Since one of the states for the 2-channel zones is already realized in the initial routing, only one other state is left to be taken. In other words, the internal loop count is a function of the number of 2-channel zones (switchable segments).

7. **The outer loop criterion:** This is the algorithm exit condition which is generally satisfied when the solution fails to improve for a certain consecutive number of times. This is done by recording the cost function value at the end of each stage of the annealing process. In our implementation, we stop when we observe no improvement in ten consecutive calls to the Metropolis function.

6.9 Results and Discussion

The two algorithms, tabu search and simulated annealing were presented in previous sections. In this section we will discuss the application of these algorithms on various benchmark circuits. The algorithms are compared with each other and the best values for their parameters that affect the performance of the algorithms are identified.

For the VLSI circuits used in our experiments, a 2μ CMOS technology has been used. The timing characteristics and other parameters such as dimensions *etc.*, are given in [49]. It is assumed that metal m_1 is used for routing horizontal tracks, whereas metal m_2 is used for vertical tracks. The values of capacitance and resistance for these metals are adopted from a fabrication foundry manual [53].

The global router is written in C language under UNIX and runs on SUN and NeXT machines. It has been tested on various practical circuits with a number of cells ranging from 50 to about 2000. Table 6.1 shows the description of the benchmarks used.

The statistics for different zones that were formed in these benchmarks while performing global routing, are tabulated in Table 6.2. Tabu search implementation has resulted in faster execution time and better results than the simulated annealing as shown in Table 6.3. The reason for this behavior is that the tabu search works in a much harder and directed way in its neighborhood to get a better solution from a set of candidate solutions (though it may not be better than the current solution), whereas simulated annealing performs almost randomly in the initial stages of its execution as the temperature is high and then settles down to obtain only good moves when the temperature reduces to zero. But in terms of memory requirements, tabu search certainly consumes much more than simulated annealing. It is to be noted however, that an iteration in tabu search is quite different from a move in simulated annealing. Tabu search performs all the steps of generation of candidate solutions, selection of a candidate solution, checking of tabu status and aspiration criteria on the particular selected solution and finally to update (or not) the current best solution, in a single iteration. On the other hand, simulated annealing makes a move to a neighboring state by simply checking the acceptance criteria defined in the Metropolis function.

As shown in Table 6.3, for the largest circuit **Struct** (1953 cells), tabu search has resulted in 10.06% reduction in cost function in just 31 seconds, whereas the SA method took 529 seconds to achieve only 2.75% improvement. Other results can be similarly noted. Figure 6.1(a) shows the performance of the tabu search procedure for the Struct circuit. The graph shows the solution cost for each iteration. Embedded on the graph is another curve for the best costs obtained. Figure 6.1(b) shows a graph for the SA case. It shows the cost against each move in the annealing process. All other graphs for other circuits can be seen from Figure 6.2 to Figure 6.6.

The timing driven router (**TDR**) is compared with the wire-mode **OASIS** placement and routing package developed by **MCNC**. Table 6.4 shows a comparison of the segment lengths in metal1 and metal2 for both TDR and OASIS.

As can be seen, OASIS produces shorter total wire lengths for most of the cases we considered. This was expected because it does not favor critical nets during global routing. Moreover, it can also be seen that the number of feed-throughs used by OASIS is quite high compared to TDR, accounting for its shorter total wire length. But, as shown in Table 6.5, for the largest circuit, Struct, the percent increase in the metal1 length of TDR with respect to OASIS is only 3.93%, where as for the metal2 length, TDR shows an improvement over OASIS for about 4.78%.

Table 6.6 shows a comparison of OASIS and TDR for the number of segments, via count, and the slack values. As shown, the Struct circuit is routed with a smaller number of segments and via count with TDR and thus resulted in better slack value. Similarly, other cases can be seen in Table 6.6.

An interesting result is shown in Table 6.7 which illustrates the effect of ordering of nets based on their criticality. As can be seen from the table, we do not have much improvement of the slack value for various smaller circuits but the net ordering becomes significant for larger circuits like Struct, whose slack value changes from 5.3 *ns* in the case of ordered nets, to -6.0 *ns* when the nets are not ordered on their criticality. This shows that the net ordering for lightly interconnected circuits has less overall effect on the routing solution (and slack values) as compared with highly interconnected designs. For dense designs with a large number of nets, competition for the routing resources (feed-throughs and tracks) becomes more severe. It is important in that case to give preference to critical nets so that they get assigned to the best feed-throughs, which results in shorter net length as well as smaller number of vias.

Circuit	Description	IOpads	Cells	Clock	Cr.Paths
Adder	Simple Adder	5	11	20 ns	6
Parl	16-bit Parity	17	15	40 ns	3
Par2	8-bit Parity	9	30	60 ns	90
TLC	Traffic Controller	10	45	18 ns	63
Fract	Fract. Multiplier	24	125	35 ns	8
Struct	16-bit Multiplier	65	1953	150 ns	90

Table 6.1: Test cases statistics.

Circuit	No. Of zones	1-chan. zones	2-chan. zones	Global zones	FT zones
Adder	23	12	9	2	0
Parl	31	12	10	9	0
Par2	59	13	38	4	3
TLC	110	49	50	8	2
Fract	315	109	142	44	19
Struct	3226	642	2306	227	51

Table 6.2: Test cases zones' statistics.

Circuits	Execution Time(sec)		Cost Reduction	
	TS	SA	TS	SA
Adder	1	5	11%	11%
Parl	1	22	12.5%	0%
Par2	1	5	14.28%	0%
TLC	1	20	16%	16%
Fract	1	49	8%	5%
Struct	31	529	10.06%	2.75%

Table 6.3: Comparison of cost reduction and execution time between SA and TS.

Circuits	OASIS			TDR		
	FTs Inserted	Length m1 μm	Length m2 μm	FTs Inserted	Length m1 μm	Length m2 μm
Adder	0	416	771	0	304	553
Par1	0	640	1313	0	664	1307
Par2	4	1968	1646	4	2408	2051
TLC	1	5336	4808	0	5656	4294
Fract	53	29608	20024	22	35254	18368
Struct	390	420136	309365	239	436672	294557

Table 6.4: Comparison of wire length between OASIS and TDR for metal1 and metal2.

Circuits	% Increase		% Decrease	
	L m1	L m2	L m1	L m2
Adder	-	-	26.92	28.2
Par1	3.75	-	-	0.457
Par2	22.3	24.6	-	-
TLC	5.9	-	-	10.6
Fract	19.06	-	-	8.2
Struct	3.93	-	-	4.78

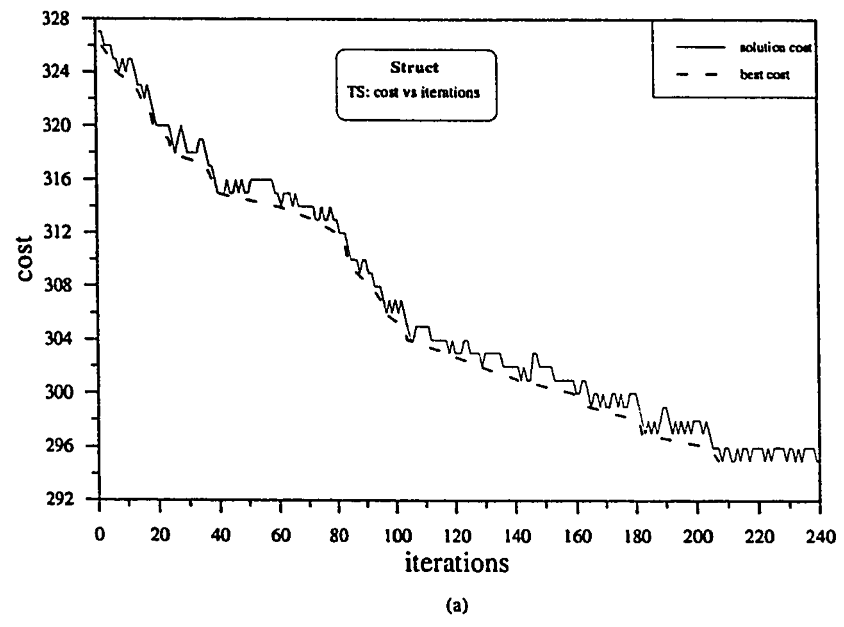
Table 6.5: Percent increase/decrease of metal1 and metal2 obtained from TDR when compared with OASIS.

Circuits	OASIS			TDR		
	Segments	Vias	Slack *ns*	Segments	Vias	Slack *ns*
Adder	22	44	2.2	19	38	2.3
Par1	39	78	5.5	38	76	5.5
Par2	61	122	1.3	61	122	1.3
TLC	119	239	10.3	114	228	10.5
Fract	390	780	4	354	708	3.5
Struct	4521	9042	4.9	4277	8554	5.3

Table 6.6: Comparison between OASIS and TDR.

Circuit	Net Order	Length m1 μm	Length m2 μm	Vias	Segments	Slack *ns*
Adder	With	304	553	38	19	2.3
	Without	384	709	48	24	2.4
Par1	With	664	1307	76	38	5.5
	Without	664	1305	76	38	5.5
Par2	With	2408	2051	122	61	1.3
	Without	2408	1447	122	61	1.7
TLC	With	5656	4294	228	114	9.5
	Without	5672	4299	234	117	9.5
Fract	With	44918	19935	708	354	3.5
	Without	32756	18335	704	352	5.2
Struct	With	436672	294557	8554	4277	5.5
	Without	505412	295205	8560	4280	-6.0

Table 6.7: Comparison between OASIS and TDR with and without net ordering.



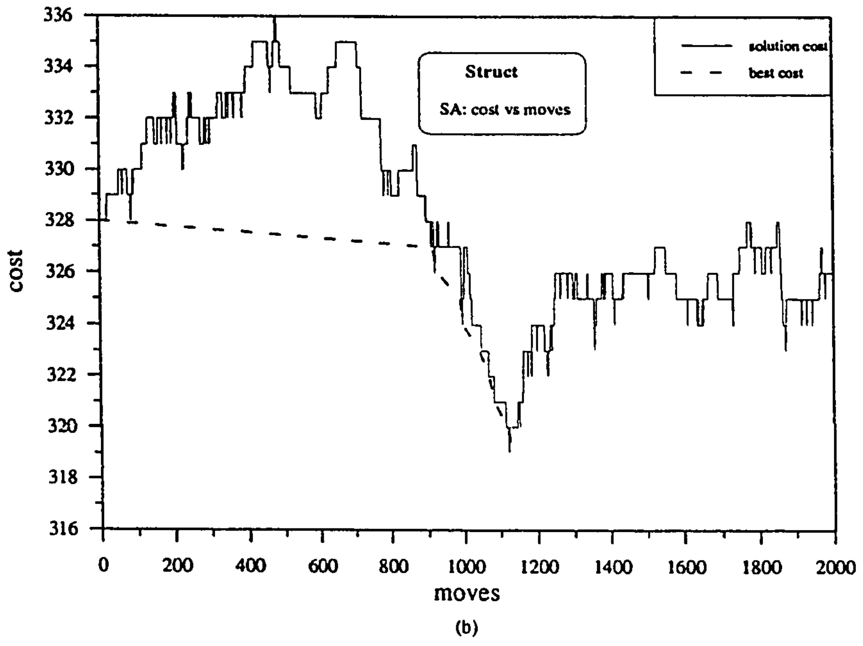
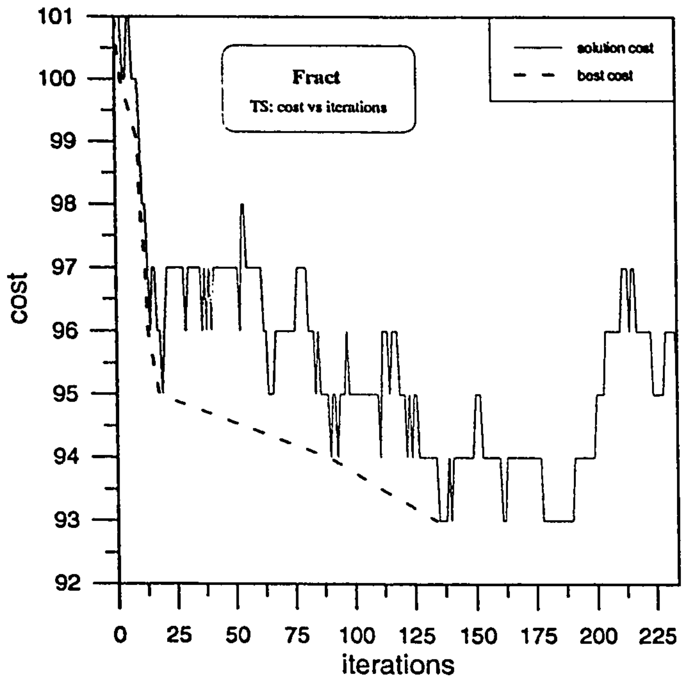
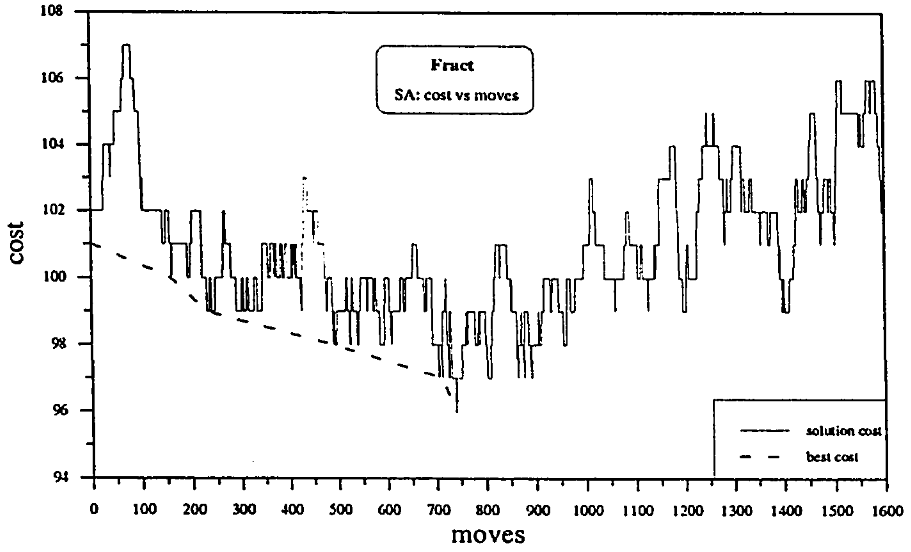


Figure 6.1: Comparison of TS and SA for the Struct circuit.

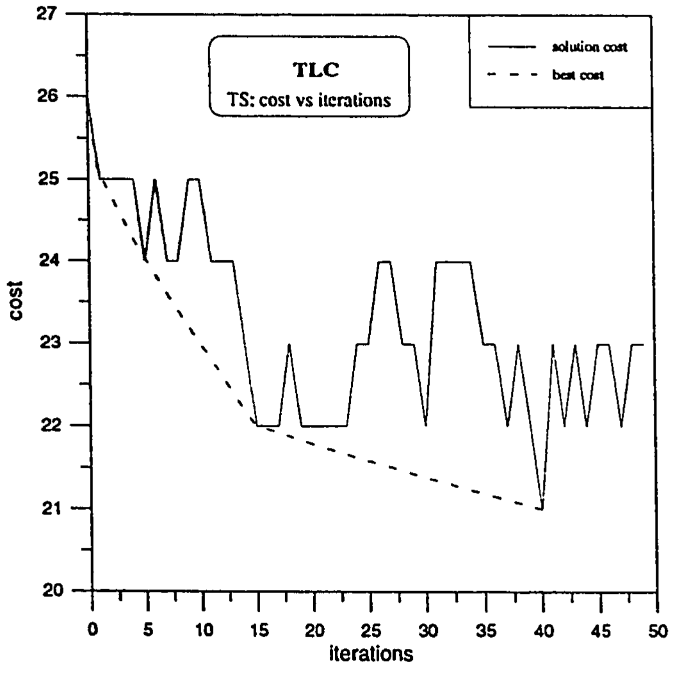


(a)

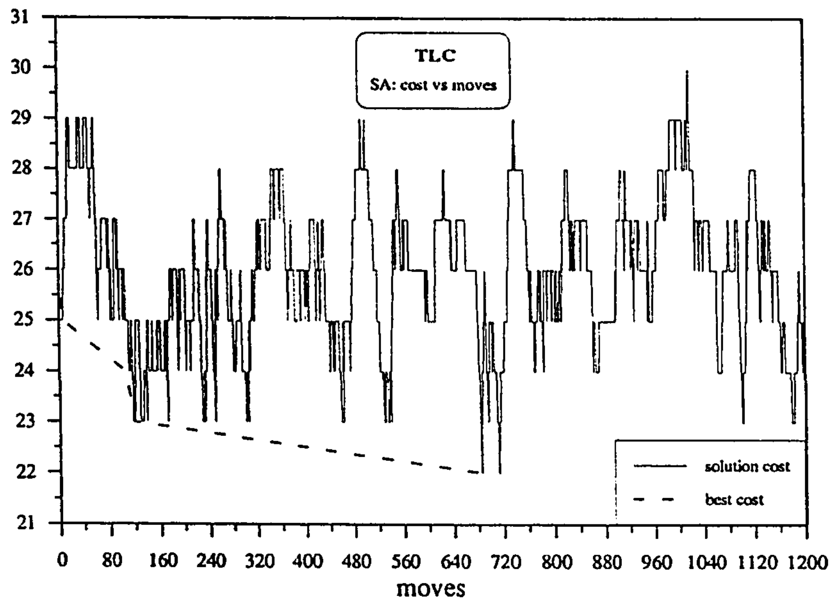


(b)

Figure 6.2: Comparison of TS and SA for the Fract circuit.

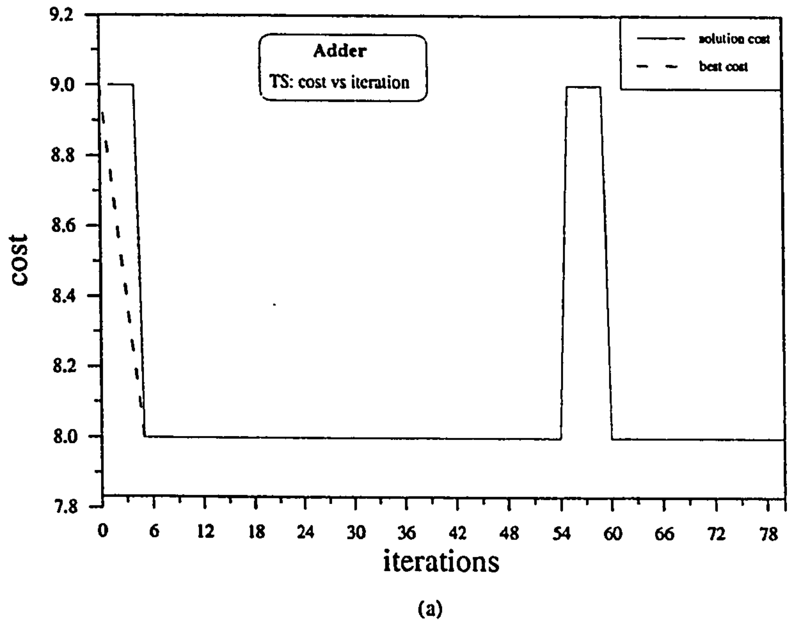


(a)



(b)

Figure 6.3: Comparison of TS and SA for the TLC circuit.



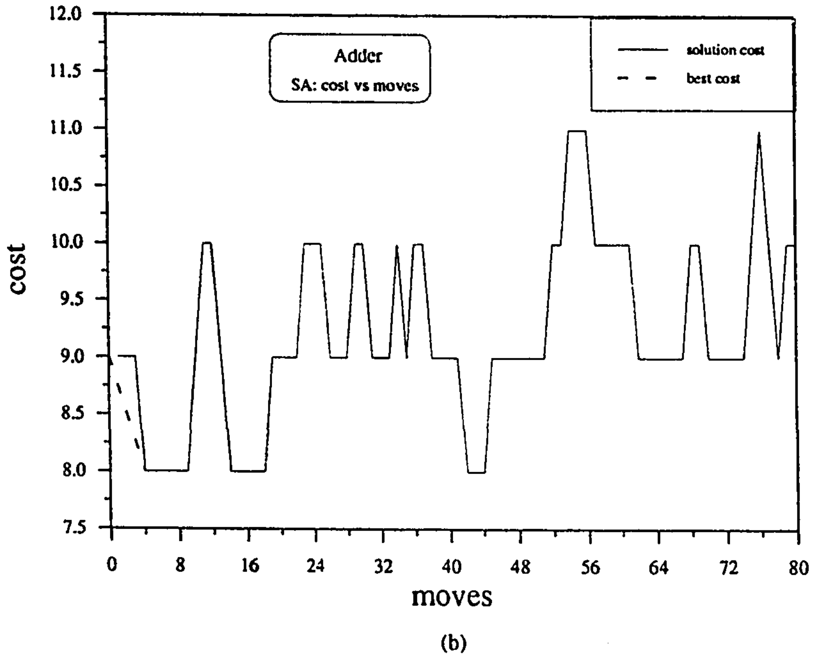
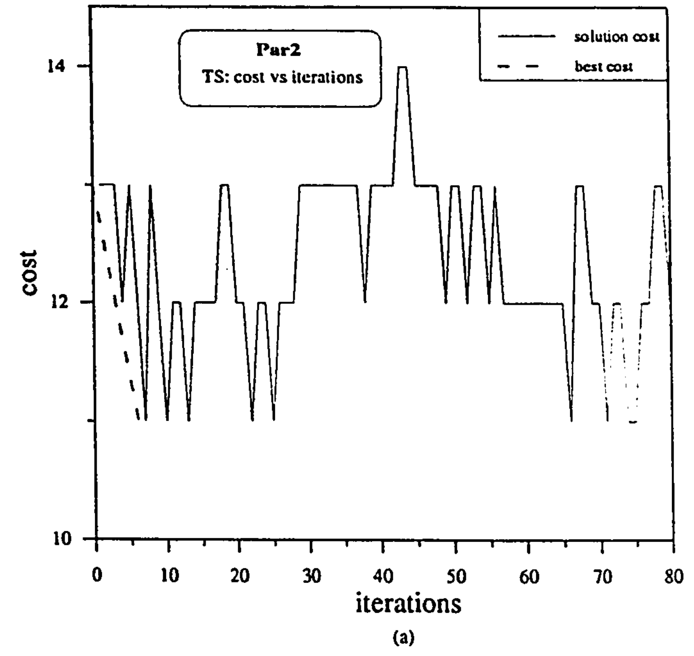


Figure 6.4: Comparison of TS and SA for the Adder circuit.



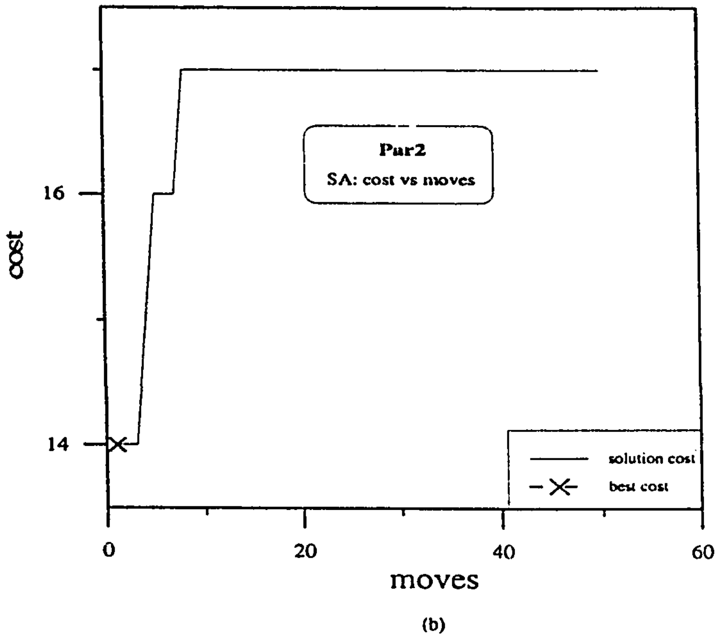
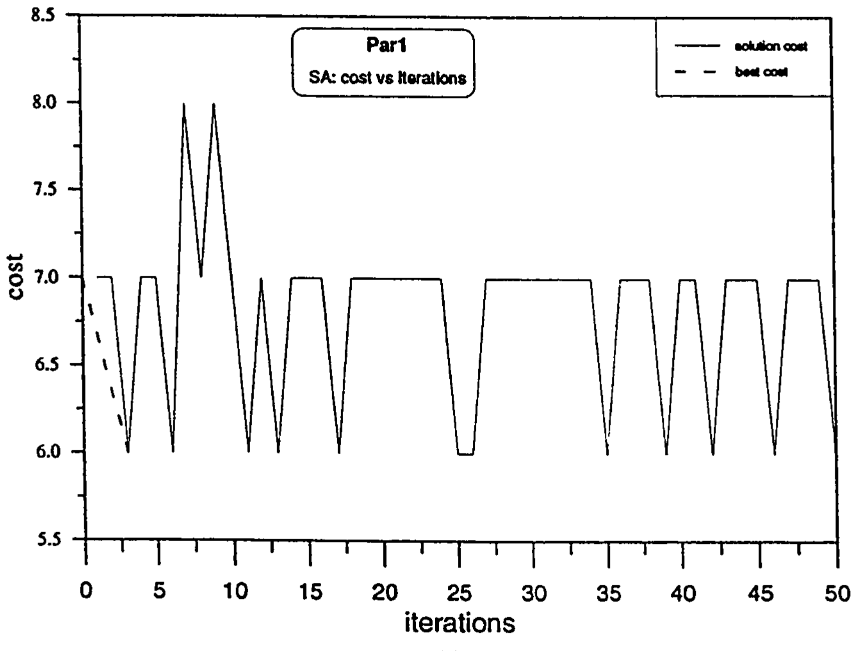
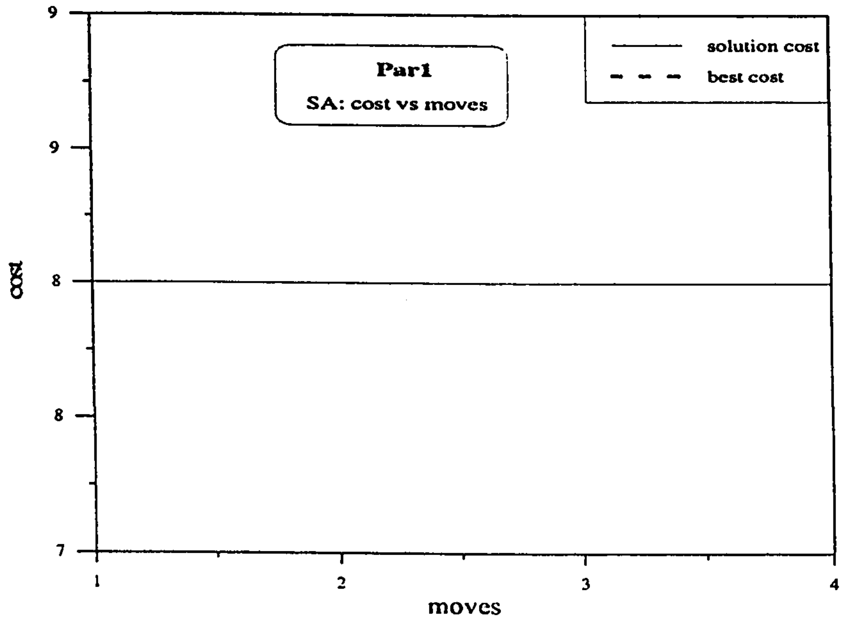


Figure 6.5: Comparison of TS and SA for the Par2 circuit.



(a)



(b)

Figure 6.6: Comparison of TS and SA for the Par1 circuit.

6.10 Summary

In this chapter, we described the iterative improvement phase of the global router where the objective is to optimize the assignment of switchable segments to reduce the overall channel densities and congestion. We also discussed some experimental results for the test cases used. In these experiments, tabu search performed much better than simulated annealing in terms of solution quality and execution time in all the test cases. Furthermore, a comparison is made between our timing driven router (TDR) and the OASIS global router in terms of wirelength, number of segments, vias and slack values.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have addressed the problem of timing driven global routing for standard cell design. A timing driven global router has been implemented. It is based on two major steps. The first stage performs the timing driven global routing in a constructive manner. The second stage improves the initial global routing solution by optimizing the assignment of switchable segments using an iterative technique called Tabu Search. The main purpose of the second stage is to minimize the overall layout area.

In Chapter 1, an overview of the global routing problem is presented, particularly in context of standard cell design. The motivation for incorporating timing during global routing phase of VLSI design process is also discussed.

In Chapter 2, a literature survey on global routing problem is given. Numerous global routing techniques are described and related work on timing driven design reported in the literature is reviewed.

In Chapter 3 an overview of rectilinear Steiner tree problem is given, which is a fundamental part of the global routing problem. Related work on Steiner tree problem is briefly discussed with special emphasis on rectilinear Steiner tree problem.

In Chapter 4, an overview of tabu search technique is given. Important characteristics of this technique are identified with an overall discussion of the tabu search algorithm. Another iterative technique called simulated annealing is also presented. This technique is also implemented for the iterative improvement stage of our initial global routing solution and the results are compared with that of tabu search as discussed in Chapter 6.

In Chapter 5, the implementation details of our timing driven global router is presented. The objective function used in the constructive phase of the global router is the timing criticality of the nets. The router is based on partitioning a net into different zones, namely one-channel, two-channel and global zones. Routing is then done inside each zone. In a one-channel zone, the terminals are in neighboring rows and the connecting segment is laid out in the channel sandwiched by the two cell rows. In a 2-channel zone, the terminals exist in the same row and therefore the segment is "Switchable". Switchable segments can be routed at random to either of the channel above or the channel below the cell row. In a global zone, the terminals are separated by one or more intermediate rows. One or more line of feed-throughs are generally needed to route the global net segments. Some of the global zones may be combined to form a feed-thorugh zone. The feed-through zone is a multi-terminal zone requiring also one or more feed-throughs.

Routing order is sequential, based on the timing criticality of the nets. Routing inside a two terminal global zone reduces to finding a shortest delay path using any shortest path algorithm. While routing inside a feed-through zone is performed by an iterative application of a shortest path algorithm to find a minimum delay tree.

The objective function used in the second stage of routing is the minimization of the layout area. This can be approximated by an optimal assignment of switchable segments. Assignment of Switchable segments (which were placed randomly in the initial routing stage) is optimized using tabu search. The results are also compared with those obtained with the Simulated Annealing technique. As reported in Chapter 6, tabu Search has resulted in good quality solutions and smaller execution times as compared to simulated annealing in all the test cases used. Also presented in Chapter 6 is a comparison between timing driven router (TDR) and OASIS [49] in terms of number of segments, vias, slack values and wire lengths for all the test cases used.

7.2 Future work

One interesting question that requires further investigation is whether net partitioning has an overall negative or positive effect on quality of solution. One way to answer this question is to find for each net a Steiner tree without partitioning the net into zones and compare the solution obtained with that produced when net partitioning approach is followed. That is, we try to build a Steiner tree for the whole net using the technique described for routing inside a feed-through zone and then use Tabu Search for improving the initial tree construction and finding a tree of minimum total delay. Similar strategy has been adopted in the TimberWolf package [13]. However, our proposed approach is different from that of Timberwolf in several important aspects: (1) Timberwolf constructs a spanning tree, while in our case we build a Steiner tree instead, (2) Timberwolf does not order the nets based on their timing criticality and does not employ timing aspects in tree construction, and (3) Timberwolf uses Simulated Annealing (with temperature set to zero) for the optimization of the assignment of the switchable segments, whereas in our approach we use Tabu Search.

Another improvement that we plan to perform is to use a more accurate delay model during the estimation of the interconnect delays. In our current implementation, we use a lumped-RC model. This model overestimates the interconnect delays. A better and more accurate model to use is the distributed-RC model.

Bibliography

[1] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice.* McGraw-Hill Book Company, Europe, 1994.

[2] Yasuo Sugai and Hironori Hirata. A global router for standard cell VLSI with feed-through assignment optimization. *Electronics and Communications in Japan*, 1990.

[3] J. T. Mowchenko and C. S. R. Ma. A new global routing algorithm for standard cell ICs. *Proceedings of 24th Design Automation Conference*, pages 27–30, 1987.

[4] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22:55–89, 1992.

[5] Fred Glover. Tabu search - part i. *ORSA Journal of computing*, 1:190–206, 1989.

[6] Takashi Kambe, Tokihito Okada, and Ikuo Nishioka. A routing scheme for standard cell VLSI. *Technical report of the institute of Electronics, Information*

and Communication Engineers, 1985.

[7] C. Y. Lee. An algorithm for path connection and its application. *IRE Transactions on Electronic Computers*, EC-10, September 1961.

[8] F. O. Hadlock. A shortest path algorithm for grid graphs. *Networks*, pages 7:323–344, 1977.

[9] J. Soukup. Fast maze router. *Proceedings of 15th Design Automation Conference*, pages 486–489, 1978.

[10] K. Mikami and K. Tabuchi. A computer program for optimal routing of printed circuit connectors. *Proceedings of IFIP*, pages H47:1475–1478, 1968.

[11] D. W. Hightower. A solution to line-routing problem on the continuous plane. *Proceedings of 6th Design Automation Workshop*, pages 1–24, 1969.

[12] M. P. Vecchi and S. Kirkpatrick. Global wiring by simulated annealing. *IEEE Transactions on Computer-Aided Design*, pages 215–222, October 1983.

[13] C. Sechen and A. L. Sangiovanni-Vincentelli. Timberwolf3.2: A new standard cell placement and global routing package. *Proceedings of 23rd Design Automation Conference*, pages 432–439, 1986.

[14] M. Burstein and R. Pelavin. Hierarchical wire routing. *IEEE Transactions on Computer-Aided Design*, pages 223–234, 1983.

[15] M. M. Sadowska. Router planner for custom chip design. *Digest of Technical papers, ICCAD*, pages 246–249, 1986.

[16] U. Lauther. Top-down hierarchical global routing for channelless gate arrays based on linear assignment. *Proceedings of IFIP, VLSI87*, pages 109–120, January 1987.

[17] M. Sarafzadeh and C. K. Wong. Hierarchical Steiner tree construction in uniform orientations. *IEEE Transactions on Computer-Aided Design*, pages 1095–1103, September 1992.

[18] Andrew Lim, Siu-Wing Cheng, and Ching ting Wu. Performance oriented rectilinear Steiner trees. *Proceedings of 30th Design Automation Conference*, pages 171–175, 1993.

[19] J. Cong. Provably good performance-driven global routing. *IEEE Transactions on Computer-Aided Design*, pages 739–752, June 1992.

[20] P. Shih, K.Chang, and W. Feng. Neural computation network for global routing. *Computer Aided Design*, pages 539–547, October 1991.

[21] S. Prasitjutrakul and W. J. Kubitz. A performance-driven global router for custom VLSI chip design. *IEEE Transactions on Computer-Aided Design*, pages 1044–1051, August 1992.

[22] Jin Huang, Xian Long Hong, Chung-Kuan Cheng, and E. S. Kuh. An efficient timing-driven global routing algorithm. *Proceedings of 30th Design Automation Conference*, pages 596–600, 1993.

[23] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. *Proceedings of 8th Design Automation Conference*, pages 155–169, 1971.

[24] A. Perskey, D. Deutch, and D. Schweikert. A system for the directed automatic design of LSI circuits. *Proceedings of 13th Design Automation Conference*, June 1976.

[25] W. K. Luk. A greedy switchbox router. *Integration, the VLSI Journal*, pages 129–149, 1985.

[26] R. L. Rivest and C. M. Fiduccia. A greedy channel router. *Proceedings of 19th Design Automation Conference*, pages 418–424, Jan 1982.

[27] M. R. Garey and D.S. Johnson. The rectilinear Steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[28] Dana Richards. Fast heuristic algorithms for rectilinear Steiner trees. *Algorithmica*, 4:191–207, 1989.

[29] R. E. Tarjan. Data structures and networks algorithms. *SIAM, Philadelphia, PA*, 1983.

[30] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal of Applied Mathematics*, 14:255–265, 1966.

[31] P. winter. Steiner problem in networks: A survey. *Networks*, 17:129–167, 1987.

[32] F. K. Hwang. On Steiner minimal trees with good performance. *SIAM Journal of Applied Mathematics*, 30(1):104–114, 1976.

[33] M. W. Bern. Two probabilistic results on rectilinear Steiner trees. *Algoritnmica*, 3:191–204, 1988.

[34] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal of Applied Mathematics*, 16:1–29, 1968.

[35] Andrew B. Kahng and Gabriel Robins. A new class of Steiner tree heuristics with good performance. *IEEE Trans. Computer Aided Design*, 11(7):893–901, JULY 1992.

[36] R. Venkateswaran and P. Matsuyama. Routing algorithms in semiconductor circuit design. *In preparation.*

[37] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Science*, 8:156–166, 1977.

[38] Fred Glover. Tabu search - part ii. *ORSA Journal of computing*, 2:4–32, 1990.

[39] A. Hertz and D. de Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1989.

[40] Andrew Lim and Yeow-Meng CHEE. Graph partitioning using tabu search. *IEEE Intl. Symp. circuit and systems*, pages 1164–1167, 1991.

[41] L. Song and A. Vinnelli. VLSI placement using tabu search. *Microelectronics Journal*, 17(5):437–445, 1992.

[42] Shawki Areibi and Anthony Vannelli. Circuit partitioning using a tabu search approach. *IEEE Intl. Symp. circuit and systems*, pages 1643–1646, 1993.

[43] C. Friden, A. Hertz, and D. de Werra. Tabaris: An exact algorithm based on tabu search for finding a maximum independent set in a graph. *Computers and Operation Research*, 19(1):81–91, 1990.

[44] Fred Glover. Artificial intelligence, heuristic frameworks and tabu search. *Managerial and decision economics*, 11:365–375, 1990.

[45] S. Kirkpatrick, C. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *IBM Computer Science/Engineering Technology Watson Res. center, Yorktown Heights, NY, Tech. Rep.*, 1982.

[46] Takeshi Fukao. Distributed system theory. *Shoukoudo*, July 1987.

[47] M. Metropolis et al. Equation of state calculation by fast computing machines. *Journal of Chemical Physics*, pages 1087–1092, 1953.

[48] J. Cong and B. Preas. A new algorithm for standard cell global routing. *Proc. IEEE Int. Conf. Computer Aided Design*, pages 176–179, Nov. 1988.

[49] OASIS: Open architecture Silicon Implementation Software. *MCNC Centre for Microelectronics*, 1990.

[50] K. Al-Farra. *Timing Driven Floorplanning (MS Thesis).* Computer Engineering Department, KFUPM, Dhahran.

[51] H. Youssef, S. Sutanthavibul, and E. Shragowitz. Pre-layout timing analysis of cell based VLSI design. *Computer Aided Design Journal*, 7:367–379, 1992.

[52] Yasuo Sugai and Hironori Hirata. A hierarchical simulated annealing method for VLSI block placement problem. *The transactions of the Institute of Electronics, Information and Communication Engineers*, pages 760–767, 1988.

[53] Foresight Manual. *Orbit Semiconductor Inc., Sunnyvale, California*, July 1992.

Vitae

- Amir Hashmi
- Born in 1968 at Karachi, Pakistan
- Received Bachelor of Engineering (B.E.) degree in Computer Systems Engineering from N.E.D. University of Engineering and Technology, Karachi, Pakistan in 1991.
- Joined the Department of Information & Computer Science at King Fahd University of Petroleum and Minerals (KFUPM), Dhahran, Saudi Arabia as a Research/Teaching Assistant in September 1992
- Received Master of Science (M.S.) degree in Computer Science from KFUPM, Saudi Arabia in 1995.