1-1-2000

Intelligent approaches to VLSI routing

Maolin Tang
Edith Cowan University

Edith Cowan University

Copyright Warning

INTELLIGENT APPROACHES TO VLSI ROUTING

By
MAOLIN TANG

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of
Doctor of Philosophy
(Computer Systems Engineering)
at
School of Engineering and Mathematics
Edith Cowan University

Principal Supervisor: Professor Kamran Eshraghian
Associate Supervisor: Dr Daryoush Habibi

2000

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

DECLARATION

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

To my wife, Lan, and my son, Chen.

Abstract

Very Large Scale Integrated-circuit (VLSI) routing involves many large-size and complex problems and most of them have been shown to be NP-hard or NP-complete. As a result, conventional approaches, which have been successfully used to handle relatively small-size routing problems, are not suitable to be used in tackling large-size routing problems because they lead to 'combinatorial explosion' in search space. Hence, there is a need for exploring more efficient routing approaches to be incorporated into today's VLSI routing system. This thesis strives to use intelligent approaches, including symbolic intelligence and computational intelligence, to solve three VLSI routing problems: *Three-Dimensional (3-D) Shortest Path Connection*, *Switchbox Routing* and *Constrained Via Minimization*.

The 3-D shortest path connection is a fundamental problem in VLSI routing. It aims to connect two terminals of a net that are distributed in a 3-D routing space subject to technological constraints and performance requirements. Aiming at increasing computation speed and decreasing storage space requirements, we present a new A^* algorithm for the 3-D shortest path connection problem in this thesis. This new A^* algorithm uses an economical representation and adopts a novel backtrace technique. It is shown that this algorithm can guarantee to find a path if one exists and the path found is the shortest one. In addition, its computation speed is fast, especially when routed nets are spare. The computational complexities of this A^* algorithm at the best case and the worst case are $O(l)$ and $O(l^3)$, respectively, where l is the shortest path length between the two terminals. Most importantly, this A^* algorithm is superior to other shortest path connection algorithms as it is economical in terms of storage space requirement, i.e., 1 bit/grid.

The switchbox routing problem aims to connect terminals at regular intervals on

the four sides of a rectangle routing region. From a computational point of view, this problem is NP-hard. Furthermore, it is extremely complicated and as the consequence no existing algorithm can guarantee to find a solution even if one exists no matter how high the complexity of the algorithm is. Previous approaches to the switchbox routing problem can be divided into *algorithmic approaches* and *knowledge-based approaches*. The algorithmic approaches are efficient in computational time, but they are unsucessful at achieving high routing completion rate, especially for some dense and complicated switchbox routing problems. On the other hand, the knowledge-based approaches can achieve high routing completion rate, but they are not efficient in computation speed. In this thesis we present a hybrid approach to the switchbox routing problem. This hybrid approach is based on a new knowledge-based routing technique, namely *synchronized routing*, and combines some efficient algorithmic routing techniques. Experimental results show it can achieve the high routing completion rate of the knowledge-based approaches and the high efficiency of the algorithmic approaches.

The constrained via minimization is an important optimization problem in VLSI routing. Its objective is to minimize the number of vias introduced in VLSI routing. From computational perspective, the constrained via minimization is NP-complete. Although for a special case where the number of wire segments splits at a via candidate is not more than three, elegant theoretical results have been obtained. For a general case in which there exist more than three wire segment splits at a via candidate few approaches have been proposed, and those approaches are only suitable for tackling some particular routing styles and are difficult or impossible to adjust to meet practical requirements. In this thesis we propose a new graph-theoretic model, namely *switching graph model*, for the constrained via minimization problem. The switching graph model can represent both grid-based and gridless routing problems, and allows arbitrary wire segments split at a via candidate. Then on the basis of the model, we present the first genetic algorithm for the constrained via minimization problem. This genetic algorithm can tackle various kinds of routing styles and be configured to meet practical constraints. Experimental results show that the genetic algorithm can find the optimal solutions for most cases in reasonable time.

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my principal supervisor, Professor K. Eshraghian and associate supervisor, Dr D. Habibi, for their encouragement and dedication. I am also grateful for the help I received from Professor Rafie Mavaddat and Dr H.N. Cheung who once served as my associate supervisors. I am also appreciative of the valuable comments on my research proposal from Associate Professor Abedesselam Bouzerdoum and the valuable comments on my thesis from Dr Amine Bermak.

I would like to thank Edith Cowan University for offering me an Overseas Postgraduate Research Scholarship and Edith Cowan University Scholarship. I would have not been able to concentrate on my study without these financial supports.

I extend my gratitude to the School of Engineering and Mathematics and the Research & Higher Degrees Office at the Faculty of Communications, Health and Science, Edith Cowan Universityy for providing me traveling grants to attend international conferences which were helpful towards my research.

Finally, I thank my wife, Lan, and my son, Chen, for their consideration over the past three years when it often appeared that I had little time for anything other than this thesis. Without their love and understanding, this thesis would have never been completed.

Publications

The followings are a list of papers published during my PhD study:

1. M. Tang, H.N. Cheung, and K. Eshraghian, "A novel shortest path connection algorithm for multilayer VLSI routing," *Proc. 3rd Biennial Engineering Mathematics and Applications Conference*, Adelaide, 1998, 479-482.

2. M. Tang, K. Eshraghian and H.N. Cheung, "A heuristic three-dimensional path connection algorithm for MCM routing," *Proc. 11th International Conference on Computers and Their Applications in Industry and Engineering*, Las Vegas, 1998, 26-29.

3. M. Tang, K. Eshraghian and H.N. Cheung, "An efficient approach to constrained via minimization for two-layer VLSI routing," *Proc. IEEE Asia and South Pacific Design Automation Conference*, Hong Kong, 1999, 149-152. (also published in SIGDA Publications on CD-ROM, ACM Press, ISBN: 0-7803-50146).

4. M. Tang, K. Eshraghian, and H.N. Cheung, "A genetic algorithm for constrained via minimization," *Proc. IEEE International Conference on Neural Information Processing*, Perth, 1999, 435-440.

5. M. Tang, K. Eshraghian, and H.N. Cheung, "A hybrid approach to switchbox routing," *Proc. 26th International Conference on Computer & Industrial Engineering*, Melbourne, 1999, 235-241.

6. M.Tang, "Artificial intelligence approaches to VLSI routing," *Proc. Inter-University Postgraduate Electrical Engineering Symposium*, Perth, 1999, 35-36.

7. M.Tang, K. Eshraghian, and D. Habibi, "Knowledge-based genetic algorithm for layer assignment," *Australasian Computer Science Conference - Australian Computer Science Communications*, Vol.23, No.1, IEEE Press, 2001. (in printing)

Contents

List of Tables

List of Figures

Chapter 1

Introduction

From early 1960's, Integrated Circuit (IC) fabrication technology has been evolved from integration of a few transistors in Small Scale Integration (SSI) to today's integration of millions of transistors in Very Large Scale Integration (VLSI). In a period of four decades, there have been four generations of ICs with the number of transistors on a single chip growing from a couple of dozens to several millions. It is expected that there will be over half a billion transistors integrated on a single chip by the year 2009 [96]. On the other hand, the requirement for the time to market has become shorter and shorter. To deal with the complexity of millions of components and to achieve a turnaround time of some two months, VLSI design tools must not only be computationally fast but they must also perform close to optimal levels required. Hence, there is a desperate need to create more effective and efficient approaches and incorporate them into VLSI design tools.

Design methods and techniques play an important role in producing sophisticated VLSI design tools to manage the increasing complexity of high quality of VLSI design with optimal speed. In order to explore more practical and effective methods and techniques for VLSI design, in this thesis we investigate intelligent approaches to VLSI routing.

In this chapter we give a background to VLSI routing, the motivation of our research, the research problems to be studied, and finally the organization of this thesis.

1.1 Background

Generally, VLSI design cycle starts with a formal *system specification* of a VLSI chip, followed by *functional design*, *logic design*, *circuit design*, *physical design*, *design verification*, *fabrication*, and eventually *packaging*. A typical design cycle can be represented by a flow chart as shown in Figure 1.1.

System specification is the first stage in VLSI design whereby the specifications of a VLSI system to be designed is constructed. The specifications include size, speed, power, and functionality of the VLSI system. Functional design follows the system specification. At this stage, the behavioral aspects of the VLSI system are considered. Logical design derives and tests the logical structure that represents the functional design. *Circuit design* develops a circuit representation based on the logical design. Physical design converts the circuit representation into a geometric representation, namely *layout*. Design verification verifies the layout to ensure that it meets the system specifications and the fabrication requirements. Fabrication prepares a wafer and deposits and diffuses various material on the wafer according to the layout description. Finally, packaging fabricates and dices the wafer.

Physical design, one of the most important stages in the VLSI design cycle, is a very complex process, and thus it is usually broken down into four sub-stages, i.e. *partitioning*, *floorplanning and placement*, *routing*, and *compaction*. The physical design cycle are shown in Figure 1.2.

In the partitioning stage, the components in the circuit representation are grouped into blocks. Many factors need to be considered in partitioning, such as, size of blocks, number of blocks and number of interconnections between the blocks. The output of partitioning is a set of blocks along with the interconnections required

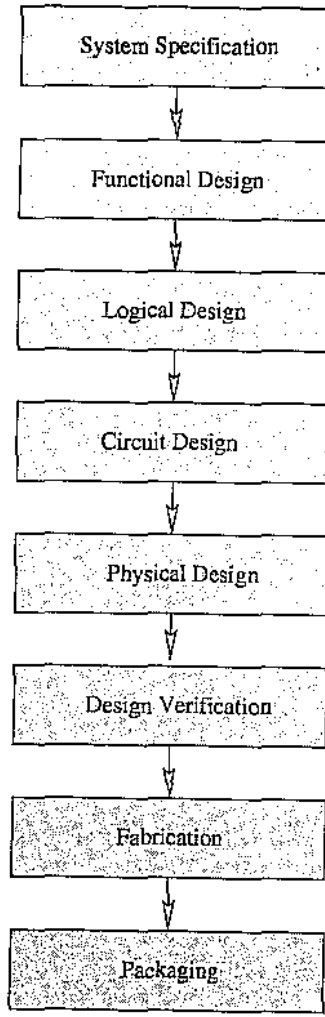


Figure 1.1: VLSI design circle

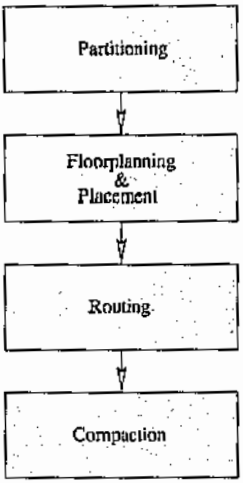


Figure 1.2: Physical design circle

between them. The set of interconnections required is referred to as the *netlist*. In the floorplanning and placement stage, a floorplanning technique is used to produce an initial layout for the blocks and a placement technique exactly positions the blocks on the chip. Then, according to the specified netlist, the routing process completes the interconnections between the blocks and produces a geometrical layout for the interconnections. Finally, in the compaction stage, the layout is compressed to reduce the total area. This thesis concentrates on investigating the problems in the routing process.

Routing is one of the most important stages of the physical design. Its goal is to find a geometric layout of all the nets in the netlist, subject to performance requirements and technological constraints. Different physical design methodologies lead to different VLSI routing problems, and different VLSI technologies require different objectives of routing. For general purpose chips the objective of routing is to minimize total wire length. For high performance chips, however, total wire length may not be a major concern. Instead, we may want to maximize their performance.

A VLSI chip may have millions of transistors and tens of thousands of nets to be routed. For each net there may be thousands of possible routes. This makes the routing problem computationally difficult. Hence, a 'divide-and-conquer' strategy is usually used to manage the high computational complexity of the routing problem. The 'divide-and-conquer' strategy partitions the routing space, the areas not occupied by the function blocks, into rectangular regions called *channel* and *switchbox*. A channel is a rectangular region with terminals at regular intervals on two opposite sides whereas a switchbox is a rectangular region with terminals at regular intervals on the four sides. Figure 1.3 illustrates channel and switchbox.

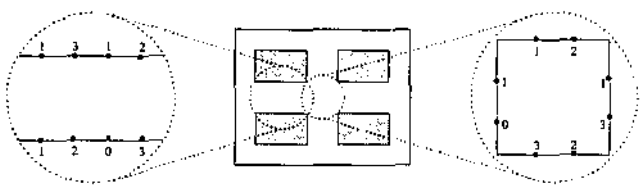
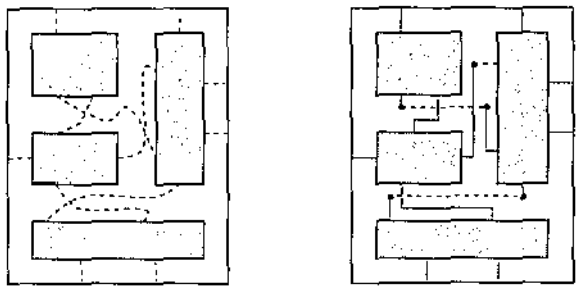


Figure 1.3: Channel and switchbox

Under the 'divide-and-conquer' strategy, VLSI routing is completed in two consecutive phases, referred as *global routing* and *detailed routing*. Global routing generates a 'loose' route for each net. In fact, it assigns a list of channels/switchboxes to each net without specifying the actual geometric layout of wires. Detailed routing finds the actual geometric layout of each net within the assigned routing regions and determines layer assignment for each wire segment of the nets. Figure 1.4(a) demonstrates global routing and Figure 1.4(b) shows a two-layer detailed routing solution based on the global routing. In Figure 1.4(b), dotted line segments represent the wire segments assigned on a routing layer, solid line segments stand for the wire segments assigned on another routing layer, and circles are *vias*. These are the conventions used by this thesis. A via is a hole (mechanism) used to connect the wire segments of a net distributed on different routing layers.



(a) (b)

Figure 1.4: Global routing and detailed routing

Although 'divide-and-conquer' decomposes a VLSI routing problem into a collection of channel and switchbox routing problems, channel and switchbox routing problems are still complex and cannot be solved in polynomial time, i.e. the channel routing is NP-complete [102] and the switchbox routing problem is NP-hard [20].

There are many design objectives and constraints, such as wire length, area, delay, crosstalk, number of vias, etc., that need to be considered in VLSI routing. Thus, it is difficult or impossible for a channel or switchbox router to produce an optimal solution in just one step. Hence, a stepwise refinement strategy is usually adopted. Under the stepwise refinement strategy, we find a feasible solution which is optimal or near-optimal in terms of a limited number of metrics. Most often we find a feasible solution with minimal length of wire segments. Then, we improve the solution by applying some post-layout optimization techniques in terms of other metrics, such as minimizing the number of vias in the solution. This research is based on the stepwise refinement strategy.

VLSI routing involves special treatment of nets, such as clock nets, power nets and ground nets. Therefore, *specialized routing* is necessary. A routing taxonomy is presented in Figure 1.5 to provide a global perspective of VLSI.

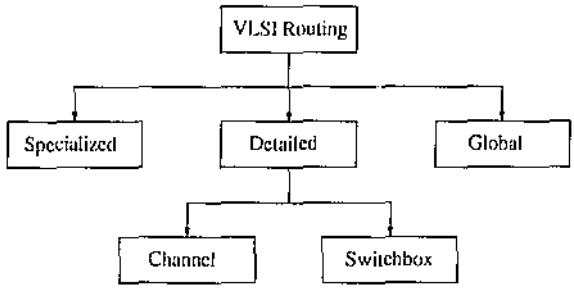


Figure 1.5: Routing taxonomy

There are two commonly used models for VLSI routing: *grid-based model* and *gridless model*. In the grid-based model, a rectilinear grid is super-imposed on the routing region. Terminals, wires and vias are required to conform to the grid. Wires are restricted to follow paths along the grid lines, while terminals and vias are forced to be at the grid points. The advantage of the grid-based model lies in the fact that the algorithms based on the grid-based model are simple. However, the grid-based model usually leads to a large amount of memory for representing the grids. Therefore, one of the challenges to grid-based algorithms is to reduce the memory requirement. In contrast, in the gridless model there are no super-imposed grids. As a result, the width of wires and the size of vias are changeable, and the location of terminals, wire segments and vias are arbitrary.

Routing models can also be divided into *unreserved routing* and *reserved routing* in terms of the method of arranging the wire segments in two-layer or multi-layer routing. If any wire segment is allowed to be placed in any layer, then the model is called unreserved layer model. If certain type of wire segments are restricted to a particular layer, then the model is called reserved layer model. In two-layer reserved layer routing, horizontal net segments are restricted to one layer and vertical wire segments are restricted to the other layer. Our research is based on grid-based unreserved layer routing model.

1.2 Motivation

VLSI has been following the Moore's Law [70] with a factor of 0.7 reduction every three years. It is expected that such reduction will continue for at least another 10-12 years according to the national technology roadmap for semiconductors in the United States of America [96]. Thus, there will be giga transistors integrated on a single chip by the end of this decade. The challenges to sustain such an exponential growth to achieve gigascale integration are shifting from the process and manufacturing technologies to design technology.

VLSI routing becomes more and more important in VLSI design automation. At low integration levels (SSI and MSI), circuit speed, packing density, and yield are determined by transistors, but as more and more devices are integrated on a single die, wires gain importance and interconnections play an important role in determining the speed, area, reliability, and yield of VLSI circuits [6].

There are at least three categories of problems challenging today's VLSI routing tools. The first category consists of the problems whose size is massive. Although their computational complexity might be polynomial, conventional approaches are still not practical because of their huge size. The second category includes problems which cannot be modeled and therefore are not suitable to be solved using conventional approaches. The third category problems are NP-hard or NP-complete problems. Although for small-size problems conventional approaches can be used very well, however this is not the case for large-size problems as the computational time and space will increase dramatically with the increase in problem size. Thus, there is a growing demand to investigate more efficient and effective approaches to VLSI routing.

Over the last four decades the field of Artificial Intelligence (AI) has significantly developed and diversified in both its conceptual advancement and applications. These advances have initiated several research areas and successfully solved many intractable problems in engineering. This research strives to use the methods and techniques of AI to solve the above three categories of problems in VLSI routing.

1.3 Research Problems

Since there are too many problems in the three categories, we cannot investigate all of them. Instead, we study intelligent approaches to VLSI routing by investigating three typical problems: *Three-Dimensional (3-D) Shortest Path Connection*, *Switchbox Routing* and *Constrained Via Minimization*, each of which belongs to one of the three categories.

The 3-D shortest path connection problem, a typical example of the first category, is a fundamental problem in VLSI routing. The objective here is to connect two terminals of a net that are distributed in a 3-D routing space subject to technological constraints and performance requirements (a multi-terminal net can be decomposed into a collection of two-terminal nets and therefore a path connection algorithm can be used for connecting the multi-terminal net). Our research on the 3-D shortest path connection problem explores a new approach which is fast in computation speed and economical in memory space.

The switchbox routing problem, an instance of the second category, is a complex problem in VLSI routing. Its objective is to connect terminals at regular intervals on the four sides of a switchbox. It is so complicated that no existing algorithm can guarantee to find a solution even if one exists no matter how high the complexity of the algorithm is. In fact, no approach has been found to determine whether a switchbox routing problem is solvable or not. Our research on the switchbox routing problem is to look at a new method capable of high routing completion rate and efficiency.

The constrained via minimization, an instance of the third category, is one of the most important optimization problems in VLSI routing. It comes after an initial routing is obtained. Its objective is to minimize the number of vias introduced in the initial routing. Our research on the constrained via minimization problem aims at finding a more robust and efficient approach to the constrained via minimization problem.

1.4 Thesis Organization

The rest of this thesis is organized as follows: In Chapter 2 we review the three research problems to be studied. Chapter 3 presents a new A^* algorithm approach to the 3-D shortest path connection problem. Here we transform the 3-D shortest path connection problem into a state-space problem of AI and then apply A^* algorithm on the state-space problem. Also a novel backtrace and clearance technique is presented. In Chapter 4 a hybrid approach to the switchbox routing problem is proposed, aiming at improving the routing completion rate and increasing the efficiency of switchbox routing. In Chapter 5, we propose a new graphic model for constrained via minimization problem and, on the basis of the model, we propose a genetic algorithm approach to the constrained via minimization problem. Finally, in Chapter 6 we conclude our research work, and outline the future research work.

Chapter 2

Review of Research Problems

This chapter reviews the three research problems which are the focus of this thesis: the 3-D shortest path connection problem, the switchbox routing problem, and the constrained via minimization problem. It investigates existing approaches to these problems and their deficiencies.

2.1 3-D Shortest Path Connection Problem

The shortest path connection problem is a fundamental problem in VLSI routing. Many routing techniques are based on the path connection algorithms, such as 'rip-up and reroute' [27]. Existing shortest path connection algorithms fall into two categories: one category consists of maze algorithms and the other line-search algorithms.

The most widely known maze algorithm is the Lee algorithm [59]. Originally, the Lee algorithm was proposed for finding the shortest path connection between two vertices on a plane grid graph. However, it can be extended to a three-dimensional grid graph. The Lee algorithm can be decomposed into two distinct phases: *exploration*, and *backtrace & clearance*. First of all, the Lee algorithm assigns either of the two grid points to be connected as 'source' grid point, say, s, and the other one

as 'target' grid point, say t. Initially, a '1' is entered in every available neighboring grid points adjacent to s. Next, a '2' is entered in every available neighboring grid points adjacent to those labeled '1', and so on. This process is continued until either t is reached or in the k^{th} step there are no available grid point adjacent to those labeled '$k-1$'. The former case indicates that there exists a path from s to t and that the length of the shortest path is k (but we do not know what the shortest path is at this stage), while the latter case means that there is no path from s to t. If t is reached, the Lee algorithm steps into the backtrace and clearance phase which is to actually find a shortest path from s to t by tracing the labeled grid points in descending order from t to s. Meanwhile, the Lee algorithm clears the labeled grid points for subsequent interconnections, except for those used for path just found.

The Lee algorithm is conceptually easy and has two excellent properties: firstly, it can guarantee to find a path connecting two points as long as one exists; secondly, if there are more than one path between the two points, the path found by the algorithm is surely the shortest one. However, the Lee algorithm is a kind of breadth-first search algorithm which belongs to the category of blind search algorithms. The search space of Lee algorithm expands rapidly as the size of problem is increased. In the worst case the computational time is of $O(N^2)$ for an $N * N$ grid graph. Moreover, the Lee algorithm requires a significant amount of memory. Low search efficiency and large storage space requirement are two major drawbacks.

In order to improve the efficiency of path connection, Hadlock [39] proposed a shortest path algorithm based on a new measure for labeling, called *detour number*. The detour number $d(P)$ of path P is defined as the total number of grid points directed away from the target grid point t. Thus, the length $l(P)$ of path P is given by

$$l(P) = M(s,t) + 2 * d(P)$$

It is clear that P is the shortest path if and only if $d(P)$ is minimized among the paths connecting s and t. Based on this idea, the exploration phase of the Lee algorithm is modified in such a way that:

1. detour numbers with respect to a specified target are entered in searched empty grid points;

2. those grid points with less detour numbers are expanded with higher priority.

The minimum detour algorithm still guarantees to find the shortest path connection. The efficiency of search is improved remarkably.

The fast maze algorithm proposed by Soukup [97] is another improvement of the Lee algorithm. In this algorithm, a line segment starting from the source grid point is initially extended toward the target grid point, and the grid points on the line segment are searched first. The line segment is extended without changing directions unless it is necessary. When the line segment hits an obstacle, the Lee algorithm is used to search around the obstacle. Once a grid point approaching the target grid is found, another line segment starting from there is extended again toward the target grid point. The fast maze algorithm again guarantees to find a path whenever it exists, but the generated path is not always the shortest one.

Another successful heuristic path connection algorithm to mention here is the global routing algorithm for general cells proposed by Clow [23]. In his algorithm, Clow [23] formulated a plane (two-dimensional, or 2-D) path connection problem into a state space problem of AI and applied the idea of A^* algorithm [75] on the state space problem. The algorithm is basically used for 2-D path connection problems and only produces a global routing solution (a detailed router is required to produce the track assignment). Although the algorithm cannot be used for 3-D path connection problems, the idea can be used to develop efficient 3-D shortest path connection algorithms. We discuss this issue in detail in Chapter 3.

In order to reduce the requirement for memory, many encoding schemes have been proposed. In the depth-first predictor search algorithm [85] each grid point uses 3 bits. Hoel [48] presented a more economical encoding scheme which uses just 2 bits for each grid point. Akers [2] also proposed a different encoding scheme that uses 2 bits for each grid point.

Line-search algorithm was first proposed by Mikami-Tabuchi [66] and Hightower [46] independently, aimed at reducing execution time and memory space requirements of the maze algorithms. Unlike the Lee algorithm and its variations in which a unit of memory space is allocated for each grid point, the line-algorithms represent routing space and paths as a set of line segments. The line-search algorithms can be viewed as proceeding on an imaginary grid. Thus, the line-search algorithms make it possible to reduce necessary memory space and execution time. In practice, both the memory space and execution time are reduced considerably for most cases. However, the line-search algorithms cannot guarantee to find a path between two points even if the path exists. In addition, the line-search algorithms cannot guarantee that the path which is found is the shortest path. The line-expansion algorithm proposed by Heyns, Sansen and Beke [45] is considered as a variation of the line-search algorithms. The so-called line-expansion algorithm is superior to the line-search in that it can guarantee to find a path if it exists. But, it still cannot find the shortest path.

Recently, some performance-driven path connection algorithms have been proposed. The time-driven maze routing algorithm proposed by Sung, Jagannathan and Lillis [101] is one of them. The algorithm adopts a multigraph model appropriate for global routing applications. Each edge in the multigraph is annotated with resistance and capacitance values associated with the particular wiring segment. The objective of the time-driven maze routing algorithm is then to find those paths which exhibit low resistance-capacitance (RC) delay or achieve a tradeoff between RC delay and total capacitance.

In summary, maze algorithms can guarantee to find the shortest path connection but they suffer from low search efficiency and large memory space requirement. In contrast, line-search algorithms achieve high search efficiency and require small memory space, but they cannot guarantee to find the shortest path connection. Furthermore, these algorithms basically focus on solving 2-D path connection problems rather than 3-D path connection problems. Moreover, most of the existing algorithm do not consider practical constraints and performance issues. Thus, there is

a need to develop a high performance 3-D path connection approach.

2.2 Switchbox Routing Problem

From computational point of view, the switchbox routing problem is considered to be a NP-hard problem [20]. The switchbox routing problem was proposed by Soukup in 1981 [98]. Since then a number of switchbox routing approaches have been proposed. The existing switchbox routing approaches to the switchbox routing problem can be divided into two categories: knowledge-based approaches and algorithmic approaches.

The most renowned knowledge-based switchbox router is WEAVER [51]. In fact, WEAVER is an expert system for both switchbox and channel routing. WEAVER consists of a number of experts (programs), each of which knows how to handle a situation in terms of a particular metric. They communicate by posting results and suggestions onto a global working memory known as a blackboard. Control is performed by a 'focus of attention' expert based on a set of priorities.

An interesting aspect of WEAVER is that, unlike human experts, it uses no backtracking. The authors felt that human backtracking decisions were too poorly understood to be coded directly into rules, and that structured backtracking techniques were inadequate and inefficient. They discovered, however, that by combining enough predictive knowledge into the planning experts, they could arrive at good solutions, better than those achievable by algorithmic approaches, with no backtracking.

WEAVER achieved very high routing completion rate. An evidence to support this is that is that it has routed Burstein's difficult switchbox automatically. WEAVER, however, is not a practical system due to the following reasons: the first and most important reason is its poor efficiency. For example, it took nearly half an hour to route the Burstein's difficult switchbox. Another reason is that WEAVER cannot be embedded into a real VLSI design system because it was implemented in OPS5

and any knowledge-based system implemented in OPS5 cannot be executed without OPS5.

Of the algorithmic approaches, BEAVER is the most successful switchbox router. BEAVER is based on a delayed layering scheme with computational geometry techniques [24]. The main objectives of BEAVER are the via and wire length minimization. BEAVER adopts the unreserved layer model routing. While routing a net, BEAVER delays the layer assignment as long as possible. One of the important features of BEAVER is that it uses priority queue to determine the order in which nets are interconnected. BEAVER uses up to three methods to find interconnections for nets: (1) corner router, (2) line router, and (3) thread router. The corner routing connects two terminals of a net that lie on adjacent sides of a switchbox and no other terminal of the net lie between the two terminals; the line sweep router produces a minimal length connection for two disjointed subnets heuristically; and the thread router is a maze router which is used to seek minimal length connections for the remaining unconnected nets. The most outstanding merit of BEAVER lies in its high computation speed. For example, it took just one second to find a solution for the Burstein's difficult switchbox.

Another successful algorithmic router is the greedy switchbox router proposed by Luk [63]. The greedy switchbox router is an extension of the greedy channel router presented in [84]. The main feature of the greedy router is that it realizes the routing of a switchbox column by column and connects net as many segments as possible in a column by using a collection of heuristic rules. The greedy switchbox router is fast and simple. However, it is often ineffective for complex switchbox routing problems because of the unidirectional plane sweeping it uses.

Based on an incremental routing strategy, a rip-up and re-route based router named MIGHTY was presented by Shin and Sangiovanni-Vincentelli [91]. It employs maze-running but has an additional feature of modifying already-routed nets. In MIGHTY, some routed nets might be ripped-up and re-routed. In addition, a cost function is used for maze routing to penalize long paths and those requiring excessive vias. MIGHTY consists of two entities: a path-finder and a path-conformer.

The worst case time complexity of the algorithm is more than $O(k^3pmL)$, where p and k are the numbers of terminals and nets, respectively, and L is the complexity of the maze algorithm.

PARALLEX is the first parallel switchbox router [20]. It can achieve a very high degree of parallelism by generating as many processes as nets, and each process is assigned to route a net. If conflicts are found for current route of a net, then that process classifies the set(s) of conflict segments into grounds that are identified by various types of conflict(s) within each group. Each process with conflicts finds partial solutions by resolving every conflict of a group in the path-finding procedure and merges them with the solutions from other processes, which may or may not have conflicts, to make a conflict-free switchbox.

Based on segment assignment in circular routing, Yan and Hsaio [118] proposed a circular assignment-based switching router, namely CONVERGE. The routing process of the CONVERGE router is divided into three phases: the iterative phase, the merging phase and the via reduction phase. In the iterative phase, circular routing is applied to assign vertical or horizontal segments cycle by cycle. In the merging phase, if switching routing is successful, the routing result is further improved by merging adjacent segments of the same net. Finally, in the via reduction phase, constrained layer assignment can further reduce the number of vias.

Aimed at satisfying crosstalk constraints on the nets and minimizing the total crosstalk among all of the nets, Tong and Liu [112] presented a minimum crosstalk switchbox routing algorithm. The proposed algorithm utilizes existing switchbox routing algorithms and improve the routing results through re-assigning the horizontal and vertical wire segments to rows and columns, respectively, in an iterative fashion.

In recent years the concepts of evolutionary computing have been applied on the switchbox routing problem. In 1996, Lienig and Thulasiraman [61] proposed a genetic algorithm for the switchbox routing problem, namely GASBOR. GASBOR is based on a three-dimensional representation of the switchbox and uses some

problem-specific genetic operators. Later on, Lienig proposed a parallel genetic algorithm for performance-driven VLSI routing, in which the length of nets, number of vias and the issues of crosstalk are considered [62]. In addition, an evolutionary program for multi-layer detailed routing (channel routing and switchbox routing) was presented by Gockel, Drechsler and Becker [35] in 1997. The evolutionary algorithm is also equipped with some domain-specific operators and can cope with up to five layer routing problems.

Although algorithmic approaches are efficient in computation speed, they are weak at achieving high routing completion rate, especially for some dense and complicated switchbox routing problems. In contrast, knowledge-based approaches are able to achieve high routing completion rate, but they are not efficient in computation speed. Therefore, there is a need to find a new approach to achieve the high computation speed of the algorithmic approaches and the high routing completion rate of the knowledge-based approaches.

2.3 Constrained Via Minimization Problem

Via minimization is one of the most important objectives in the detailed routing. There are two different approaches to minimize the number of vias. One is *Constrained Via Minimization* and the other is *Unconstrained Via Minimization*. In the constrained via minimization problem, the topology of the routing solution is fixed. The number of vias is minimized by reassigning the net segments of a routing to the available layers. On the other hand, in the unconstrained via minimization problem, the objective is to find a routing topology with minimum number of vias. Generally speaking, the unconstrained via minimization is more effective than the constrained via minimization in reducing the number of vias. However, the unconstrained via minimization cannot be used in an incremental routing as it may affect other metrics of a routing. As a consequence, constrained via minimization problem is widely used in real VLSI routing systems.

The constrained via minimization problem originated in the pioneer PCB design work of Hashimoto and Stevens [43] in 1971. It was initially formulated as a graph-theoretic max-cut problem. In their formulation, they assume that the layout is grid-based and all nets contain exactly two terminals. Later, Stevens and Van-Cleemput [100] proposed a similar but more general graph model for the same assumptions. Since the problem was thought to be NP-complete, many researchers developed heuristic algorithms [100] or used integer programming with branch and bound techniques to obtain an optimum solution [22].

In 1980, Kajitani [54] identified the net segment clusters in a layout and showed that the graph in Hashimoto and Stevens' model is planar. He further showed that the two-layer constrained via minimization problem can be solved in polynomial time when the routing is restricted to a grid-based model, and all the nets are two terminal nets. Kajitani's results encouraged other researchers to look for polynomial time algorithms for more general cases. In 1982, Pinter [80] proposed an optimal algorithm for two-layer constrained via minimization problem when maximum junction degree is limited to three. But the algorithm cannot be applied to layouts which contain junctions with higher degrees. Independently, Chen, Kajitani, and Chan [11] presented a polynomial time algorithm for grid based layout which gives optimum result when the maximum junction degree is limited to three. In 1997, Shi [94] presented a linear (non-integral) programming approach to the two-layer constrained via minimization problem. This approach can find optimum solutions for the constrained via minimization problems whose maximum junction degree is limited to three. The computational complexity of the linear programming approach is also polynomial. The other elegant theoretical results for the same situation can be found in [58, 5]. However, a general constrained via minimization problem is NP-complete [71, 14, 1].

In order to solve more general constrained via minimization problems, many approaches have been proposed. Xiong and Kuh [116] formulated the two-layer constrained via minimization problem as a $\{0,1\}$ linear programming problem. The proposed heuristic algorithm based on the model can handle the situation where

conjunction degree is greater than three.

In addition to the two-layer constrained via minimization algorithms, there are a number of multi-layer constrained via minimization algorithms proposed. In 1988, Chang and Hu showed that three-layer constrained via minimization is NP-complete, and proposed a heuristic algorithm for three-layer constrained via minimization problems. Takahashi and Watanabe [103] also proposed a heuristic algorithm for constrained via minimization problems based on the technique of the breadth-first search. Chou and Lin [21] transformed the multi-layer constrained via minimization problem into a constrained k-way graph partitioning problem, and used a modified simulated annealing program to solve the constrained k-way graph partitioning problem. The proposed approach can consider some practical issues, such as pin-out constraint, over-the-cell constraint, and overlapping between wire segments of the same net. Chang and Cong [16] formulated the multi-layer layer assignment problem in the notion of extended conflict continuation (ECC) graph. When the formulated ECC graph of a layer assignment problem is a tree, they showed that the problem can be solved in linear time. When the formulated ECC graph is not a tree, they constructed a sequence of maximal induced subtrees from the ECC graph, and then applied the linear time algorithm to each of the induced subtrees to refine the layer assignment such that the number of vias is minimized. Recently, Chuan-Jin Shi [95] applied a newly introduced graph-theoretic notion of signed hypergraph on the constrained via minimization problem. The proposed model transforms the constrained via minimization problem into the maximum balance problem of a signed hypergraph, and can represent routings containing junctions with higher degrees. A linear-time heuristic algorithm for solving the maximum balance problem is proposed.

Despite such developments, the research on constrained via minimization problem is still open. This is due to the following reasons: firstly, most algorithms cannot handle the situation where there are more than three wire segments split at a via candidate. However, in almost every real routing instance, four or more wire segments at a via candidate cannot be avoided. Secondly, although some

approaches consider the condition that there are more than three segments split at a via candidate, they depend on initial routing algorithm and cannot handle irrational situations that might be introduced by manual work. For example, the Xiong and Kuh's approach can only be used for Manhattan routing or knock-knee routing. Finally, most of these approaches are difficult to implement and therefore have not been incorporated into VLSI design tools.

2.4 Summary

In this chapter we reviewed three research problems: the 3-D shortest path connection problem, the switchbox routing problem and the via constrained via minimization problem. The discussion concentrated on existing approaches and their pros and cons. The next three chapters discuss how to use the methods and techniques of AI to solve these problems. Chapter 3 presents an A^* algorithm approach to the shortest path connection problem; Chapter 4 discusses a hybrid approach to the switchbox routing problem; and Chapter 5 proposes a genetic algorithm approach to the constrained via minimization problem.

Chapter 3

A^* Algorithm Approach to 3-D Shortest Path Connection Problem

3-D shortest path connection is a fundamental problem in multi-layer VLSI routing. The concept is to connect two terminals of a net in a 3-D routing space (a multi-terminal net can be decomposed into a set of two-terminal nets and therefore a path connection algorithm can be used for connecting the multi-terimal net), and it is widely used in global routing and detailed routing. A good path connection algorithm contributes to a high quality routing.

From computational point of view, the 3-D path connection problem is just a problem of polynomial complexity. However, with the advance of manufacturing technology, the size of path connection problem is increasing dramatically due to the constantly decreasing routing spacing and more the increasing routing layers. The number of grids becomes so large that routing programs either run out of memory or become very slow due to excessive paging [99]. For example, if we represent the information needed for one grid in an integer (4 bytes for a 32-bit machine), for a $20mm \times 20mm$ VLSI chip with routing spacing of $0.5\mu m$ and 3 routing layers we need $3 \times 40,000 \times 40,000 = 4,800,000,000$ grids, or $19.2GB$.

Moreover, in order to run the programs, we need even more memory to store netlist, routing geometry and other data. This requires large amount of memory generally not available on a personal computer or a workstation. Moreover, on larger workstations excessive and erratic paging slows down the programs so much that it becomes impractical to use. In addition, large routing space will result in large search space and therefore lead to slow computation. Thus, there is in desperate demand for investigating more efficient path connection algorithms which are fast in computational time and low in storage space requirement.

Aiming at increasing computation speed and reducing the requirement for memory space, we propose a new approach to the 3-D shortest path connection problem. This new approach transforms the 3-D shortest path connection problem into a state space problem of AI. Then a modified A^* *algorithm* is applied on the state space problem. Due to the power of A^* algorithm the search space is very small and thus the computation speed of this new approach is very fast. Besides, this new approach uses a more economical representation, i.e. 1 bit per grid point. This chapter details this A^* algorithm approach to the shortest path connection problem.

3.1 Problem Formulation

It is known that the major constraints for VLSI routing are the minimum width of wires w and the minimum clearance between any adjacent wires c. Assuming that the two values are fixed throughout all wires, we can combine the constraints into a central-to-central constraint between wires, namely, *routing spacing*, denoted as d, where $d = w + c$. Hence, the multi-layer routing environment can be modeled as a 3-D grid graph shown in Figure 3.1. The length of edges is a unit length, d. By forcing the terminals of nets to be arranged at the grid points of the graph and forcing the wire routed along the edges, the width of wire and the clearance between wires are satisfied automatically.

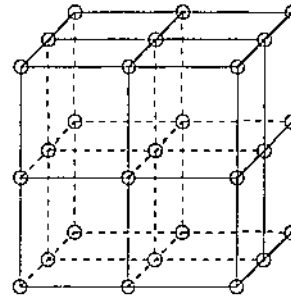


Figure 3.1: The abstract model of multi-layer routing

It should be pointed out that in a multi-layer routing environment there could be some routed nets when we find the shortest path for a pair of terminals. Reflected on the corresponding grid graph, those routed nets are a collection of grid points and edges which cannot be used for connecting this pair of terminals. We call those unavailable gird points and edges *obstacles*. Thus, the 3-D path connection problem for multi-layer VLSI routing is formulated as:

Given a 3-D grid graph and two grid points s and t, find a path connection between s and t within the 3-D grid graph in which there may be some obstacles such that the length of the path is minimum.

3.2 Overview of A^* Algorithm

A^* Algorithm is an intelligent search method of AI which is used for solving *state space problems* [75]. A state space problem is a problem which is represented as an initial state s, a set of goal states G and a collection of operators O that define operations to migrate from one state to another. The states that can be reached from s by applying those operators in all possible ways define *state space* [4]. The state space problem is then to reach one of the goal states in G from the initial state s.

A^* algorithm is a kind of heuristic algorithm which employs an additive evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the currently path from s to n and h is a heuristic estimate of the cost of the path from n to a goal state and it must be a lower bound to the cost of any path from n to the goal state. Starting from s, A^* algorithm applies all the operators in O on the current state to generate all successors of the given state and always selects the successor which has the smallest f to expand. The search halts as soon as either a goal state is reached or no new state can be explored. The former situation indicates that a path between s and t exists and the latter situation means that no path exists between s and t. A^* algorithm is described as below:

Algorithm 3.1 A^* **Algorithm**

1. *put the start state s on a list called OPEN and compute* $f(s)$*;*
2. *if OPEN is empty then*
3. *exit with failure;*
4. *remove from OPEN a node, n, whose f value is the smallest and put it on a list called CLOSED;*
5. *if n is a goal node then*
6. *exit with the solution path obtained by tracing back through the*
7. *predecessor pointers;*
8. *expand node n, generating all of its successors;*
9. *put the successors that are not on either OPEN or CLOSED on OPEN, direct pointers from them back to n, and compute f of them;*
10. *go to 2.*

In the A^* algorithm, $OPEN$ and $CLOSED$ are two linear lists which are used to retain the frontier states and explored states, respectively. It has been proven that an A^* algorithm is *admissible*, that is, it is guaranteed to find the optimal path from the initial state s to a goal state in G if one exists [75]. This is the merit of

the A^* algorithm over the other heuristic algorithms.

3.3 State Space Representation for the 3-D Path Connection Problem

In order to apply the A^* algorithm method to a practical problem, one must transform the problem into a state space problem and construct an evaluation function which meets some particular requirements. This section details how to transform a 3-D path connection problems into a state space problem. The construction of the evaluation function will be discussed in the following section.

Based on the multi-layer VLSI routing model, the 3-D path connection problem can be transformed into a state space problem $P = (s, O, G)$, where s is the initial grid point, $G = \{t\}$ is the goal state set (we assign either of the two grid points as the initial state and the other as the goal state), and $O = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ is an set of operators defined as follows:

$o_1(x, y, z) = (x - 1, y, z)$ if the grid point $(x - 1, y, z)$ is available and $x > 1$

$o_2(x, y, z) = (x + 1, y, z)$ if the grid point $(x + 1, y, z)$ is available and $x < m$

$o_3(x, y, z) = (x, y - 1, z)$ if the grid point $(x, y - 1, z)$ is available and $y > 1$

$o_4(x, y, z) = (x, y + 1, z)$ if the grid point $(x, y + 1, z)$ is available and $y < n$

$o_5(x, y, z) = (x, y, z - 1)$ if the grid point $(x, y, z - 1)$ is available and $z > 1$

$o_6(x, y, z) = (x, y, z + 1)$ if the grid point $(x, y, z + 1)$ is available and $z < p$

where (x, y, z) is a state of the state space corresponding to a grid point in the 3-D grid graph, and the size of the 3-D grid graph is $m \times n \times p$. A solution to the state-space problem is a finite sequence of applications of the operators that migrates from the initial state s into the goal state t. In this way, a 3-D path connection problem can be transformed into a state space problem.

3.4 Design and Analysis of Our A^* Algorithm

Like other maze path connection algorithms, this algorithm consists of the two consequent phases: *Exploration* and *Backtrace and Clearance*. In the first phase, the algorithm starts from the initial state s to search the goal state t. Once t is reached, the algorithm enters the second phase, that is, to trace back the shortest path among the states that have been traversed and to clear the marks made during the exploration. In contrast to other path connection algorithms, this algorithm adopts A^* search technique of AI to guide the search process and uses a novel backtrace method to retrace the shortest path connection.

3.4.1 Representation of the state space problem

The state space problem $P = (s, O, t)$ is represented as a 3-D binary array $M(1 : m, 1 : n, 1 : p)$. An element of M, $M[x, y, z]$, corresponds to the state $< x, y, z >$ in the state space defined by P, i.e. '0' stands for the state that has not been reached and '1' indicates either the state that has been explored during the exploration or the corresponding grid point has been occupied by other nets' connection.

Note that in this routing representation we do not keep any backtrace information, such as predecessor pointer, at state nodes. This is because this algorithm uses a novel backtrace technique which can identify the predecessor state of a state from the information associated with the state. As a result, we do not need a great amount of memory to be used for storing the backtrace information for all the state nodes, most of which are not used in a path connection problem. Details will be presented shortly in this section. It can be seen that from this representation that we use only 1 bit for a grid point. It is obvious that it is the most economical representation for grid-based routing.

3.4.2 OPEN and CLOSED

In addition to the 3-D binary array M used for representing the state space problem, two auxiliary linear lists are used in this algorithm: OPEN and CLOSED. The linear list OPEN is used to store all the current frontier states and the linear list CLOSED is used to store all explored states which are used for backtrace. The logical structures of OPEN and CLOSED are shown in Figure 3.2 and Figure 3.3 respectively.

$< n_1 >$	$< n_2 >$	...	$< n_i >$	...
$g(n_1)$	$g(n_2)$	...	$g(n_i)$	...
$h(n_1)$	$h(n_2)$	...	$h(n_i)$	...

Figure 3.2: The logical structure of OPEN

$< n_1 >$	$< n_2 >$	...	$< n_i >$	...
$f(n_1)$	$f(n_2)$	...	$f(n_i)$	...

Figure 3.3: The logical structure of CLOSED

In the above figures, the parameter $< n_i >$ stands for the frontier state n_i, the parameter $f(n_i)$ is the value of the evaluation function for the state n_i, the parameter $g(n_i)$ is the value of the actual cost of the path from s to n_i , and the parameter $h(n_i)$ is the value of the predicated cost of the path from n_i to t, where $i = 1, 2, \cdots$. In order to increase the efficiency of the exploration we always put a frontier state node into OPEN at its head and put an explored state into CLOSED at its tear. When selecting a frontier state in the exploration, we always scan the linear list OPEN from the head to the tail. When backtracking a shortest path in the backtace and clearance, we always scan the linear list CLOSED from the tail to the head.

3.4.3 Exploration

It is well-known that the key problem to enhance the efficiency of a shortest path connection algorithm is how to reduce search space in the state space since the

computational time of a path connection algorithm is proportional to its search space. The exploration is an ordered search process which is guided by the evaluation function defined in Equation 3.1. The evaluation function is a measurement to judge which frontier state is more promising than the others.

$$f(n) = g(n) + h(n) \tag{3.1}$$

In this equation n is a frontier state, $g(n)$ is the length of the shortest path connecting the initial state s and the frontier state n and $h(n)$ is an estimation of the length of the shortest path connecting n and the goal state t.

To derive the functions $g(n)$ and $f(n)$ we assume that n_p is the predecessor state of n, that is, n_p was the state through which n was reached, and that $s =< x_s, y_s, z_s >$, $t =< x_t, y_t, z_t >$, $n =< x, y, z >$, and $n_p =< x_p, y_p, z_p >$, where $1 \leq x_s, x_t, x, x_p \leq m$, $1 \leq y_s, y_t, y, y_p \leq n$, $1 \leq z_s, z_t, z, z_p \leq p$. The function $g(n)$ can be computed by Equation 3.2 and the function $h(n)$ is computed by Equation 3.3.

$$g(n) = \begin{cases} 0 & \text{if } n = s \\ g(n_p) + 1 & \text{otherwise} \end{cases} \tag{3.2}$$

$$h(n) = |x_t - x| + |y_t - y| + |z_t - z| \tag{3.3}$$

It can be seen that the function $f(n)$ works as an estimate of the length of the partly completed path connecting s and t and passing through the state n. In fact, it is a lower bound of the partly completed path whose frontier is n. In addition, it is not difficult to observe that $h(n)$ is a lower bound of the shortest path connecting n and t. Therefore, this evaluation function satisfies the properties of A^* algorithm and thus this path connection algorithm is admissible. Since the objective of this algorithm is to find a shortest path connecting s and t, this algorithm always gives priority to the state with minimal f to expand. When there are more than one frontier states whose f are the minimum, we choose the state whose h is minimum

among them to expand. This is due to the fact the frontier state with minimal h among them is the nearest one to the goal state t, and therefore it is expected that from this frontier state the goal state t could be reached quickly.

Starting from the initial state s, the exploration is to search the goal state t by exploring the frontier states orderly according to the dynamically generated evaluation information. The exploration is a loop of 'selecting-and-expanding'. By selecting we mean selecting a frontier state from OPEN and by expanding we refer to producing all the unexplored neighbour states of a state. Initially, the only frontier state is s, and no state has been explored. Therefore we put s together with its $g(n)$ and $h(n)$ into OPEN, and leave CLOSED empty. Then, we select a frontier state n from OPEN according to the evaluation information. We check all the unexplored neighbour states of n one by one to see if they are the goal state t. If the goal state is found, then the exploration terminates and the control is shifted to the procedure of backtrace and clearance; If it is not found, then we insert it, together with $g(n)$ and $h(n)$, into OPEN. This is so-called expanding. After expanding we move n from OPEN to CLOSED. This is a round of 'selecting-and-expanding'. The process of 'selecting-and-expanding' is repeated until either t is reached or OPEN is empty. If the latter situation occurs then no path exists between s and t.

It can been seen that every node, n, in OPEN corresponds to a partly completed path connection from s to t passing through the frontier state n. On the path, the part from s to n has been completed, while the part from n to t has not yet been completed. The course of the exploration is illustrated in Figure 3.4. In that figure, circles filled with black represent explored states, unfilled circles stand for unexplored states, and circles with shadow are frontier states. The part consisting of solid lines is the completed part of the candidate path while the part consisting of broken lines is the uncompleted part of the candidate path. The frontier state n is the expanding frontier state. Figure 3.4(a) and Figure 3.4(b) are the situations before and after expanding the frontier state n, respectively. The algorithm for the exploration is described below:

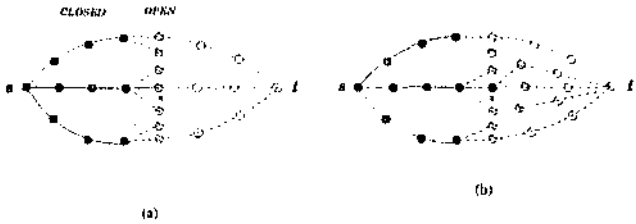


Figure 3.4: Exploration

Algorithm 3.2 Exploration

1. *put initial state s, together with $g(s)$ and $h(s)$, into OPEN;*
2. *if $OPEN = \phi$ then*
3. *exit with failure;*
4. *select a frontier state n from OPEN according to the heuristic information;*
5. *if $n = t$ then*
6. *exit with success;*
7. *$M(n) = 1$;*
8. *calculate $f(n)$;*
9. *remove n from OPEN and put it, together with $f(n)$, onto CLOSED;*
10. *expand n to find new frontier states and for each of the frontier states, q:*
    - *(a) calculate $g(q)$;*
    - *(b) calculate $h(q)$;*
    - *(c) push q together $g(n)$ and $h(q)$ onto OPEN.*
11. *go to 2.*

The exploration algorithm can be better understood by demonstrating the process

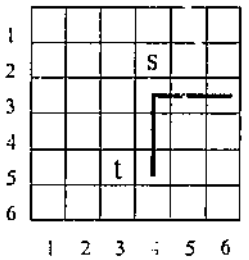


Figure 3.5: An instance of the shortest path connection problem

for an instance. For simplicity, we use an instance of the path connection problems in single-layer routing. For the 3-D path connection problems the idea is just the same. In addition, for the convenience of presentation we use the *dual graph* of the grid graph to represent the path connection problem. On the dual graph the terminals located at the grid cells rather than at the grid points. Figure 3.5 shows an instance of the shortest path connection problem and Figure 3.6 shows the process of the exploration.

In Figure 3.6, the column 'State Space' displays the status of the state space problem, the columns 'OPEN' and 'CLOSED' reflect the status of OPEN and CLOSED respectively, the column 'n' is the state that is selected to expend, and column $f(n)$ is the value of the evaluation function for n. Each row in this figure corresponds to an intermediate status of a round of 'selecting-and-expanding'. If state n is selected to expand in round i, for example, the status after the expending is displayed in the following row. For this problem the exploration takes four rounds to get to the goal state $t =< 5, 3 >$ from the initial state $t =< 2, 4 >$. The value of $f(t)$ indicates the length of the shortest path. For this instance, the length of the shortest path connection is 4.

Please note that if t is reached it means that there exists a path between s and t but we still do not know what the shortest path is. Hence we need to backtrack the shortest path traversed and clear the marks used during the exploration.

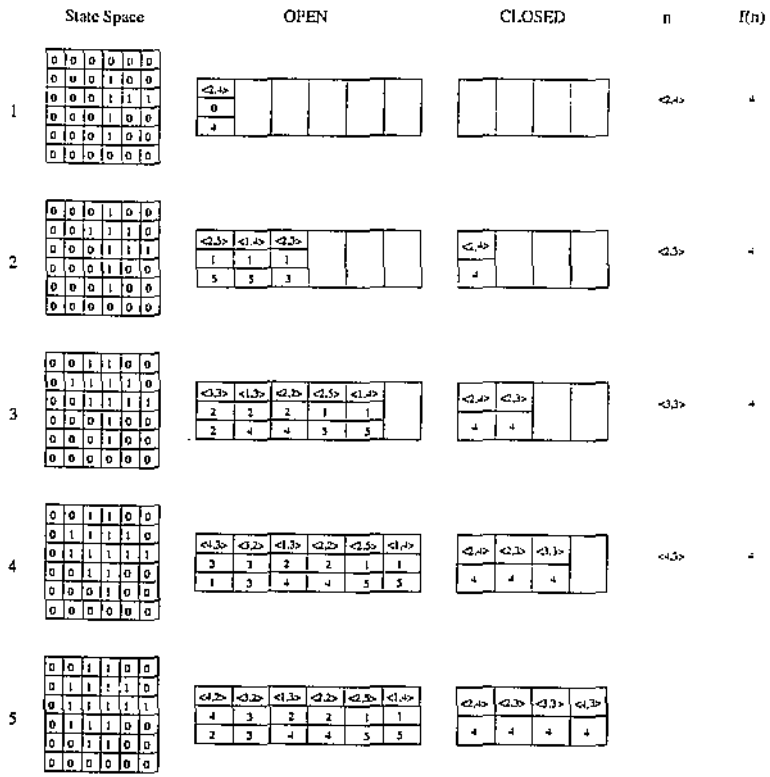


Figure 3.6: The process of exploration

3.4.4 Backtrace and Clearance

Once the target grid point t is reached, the algorithm enters the phase of backtrace and clearance. The task of the backtrace and clearance is to find the shortest path traversed during the exploration and clear the marks used for the next path connection.

To the best of our knowledge, all other existing algorithms for shortest path connection have to keep the backtrace information at explored grid points. Therefore at each grid point we must allocate some memories for retaining the backtrace information. As a result a large amount of memories need to be used. In contrast to the other algorithms, this algorithm does not need to store any predecessor pointers because it backtracks the shortest path by computing instead of utilizing the predecessor pointers.

This backtrace technique is based the following observations: first, the predecessor state of n must be one of the six neighboring states and stored in CLOSED; second, the value of $f(n)$ of the predecessor state of n equals either $f(n)$ or $f(n)-2$. Therefore, starting from t and terminating at s, by using such a technique the algorithm can backtrack the shortest path traversed during the exploration. To clear the marks used in the exploration the algorithm simply restores the states which are stored in OPEN and the states which are stored in CLOSED but not on the shortest path connection to '0'. The procedure of the backtrace and clearance is shown in Algorithm 3.3.

Figure 3.7 illustrates the process of the backtrace and clearance for the path connection problem shown in Figure 3.5. Starting from the goal state $t =< 5,3 >$, the backtrace and clearance finds its predecessor $< 4,3 >$ since $f(< 4,3 >) = f(t) = 4$. Similarly, it can find the predecessor of $< 4,3 >$, and so on and so forth, until the initial state s is reached. In this way the shortest path is found. The arrows in the figure shows the tracks of the backtrace and clearance. The status of the state space shown in the 4^{th} row is the situation when the shortest path has been found, and the status of the state space shown in the 5^{th} row is the situation after the

marks have been cleared.

Algorithm 3.3 Backtrace and Clearance

1. $q = t$;
2. *while* $q \neq s$ *do*
   (a) *find a neighboring state* p *of* q *in CLOSED such that* $f(p) = f(q)$ *or* $f(p) = f(q) - 2$;
   (b) *output* p;
   (c) *remove* p *from CLOSED*;
   (d) $q = p$;
3. *for* $\forall n \in OPEN$ *or* $CLOSED$, $M(n) = 0$;
4. *exit with success*;

3.4.5 Algorithm analysis

This proposed A^* algorithm has the following properties:

Property 3.1. The algorithm can guarantee to find a path connection between s and t if one exists.

Proof. From the algorithm we know that all frontier states are stored in OPEN at any time during the exploration, and that the algorithm will not terminate until either the goal state t is reached or OPEN is empty. So, the goal state t must be reached as long as it is reachable from the initial state s. This means that the algorithm can guarantee to find a path connecting s and t if one exists. □

Property 3.2. The algorithm can guarantee to find the shortest path connection between s and t.

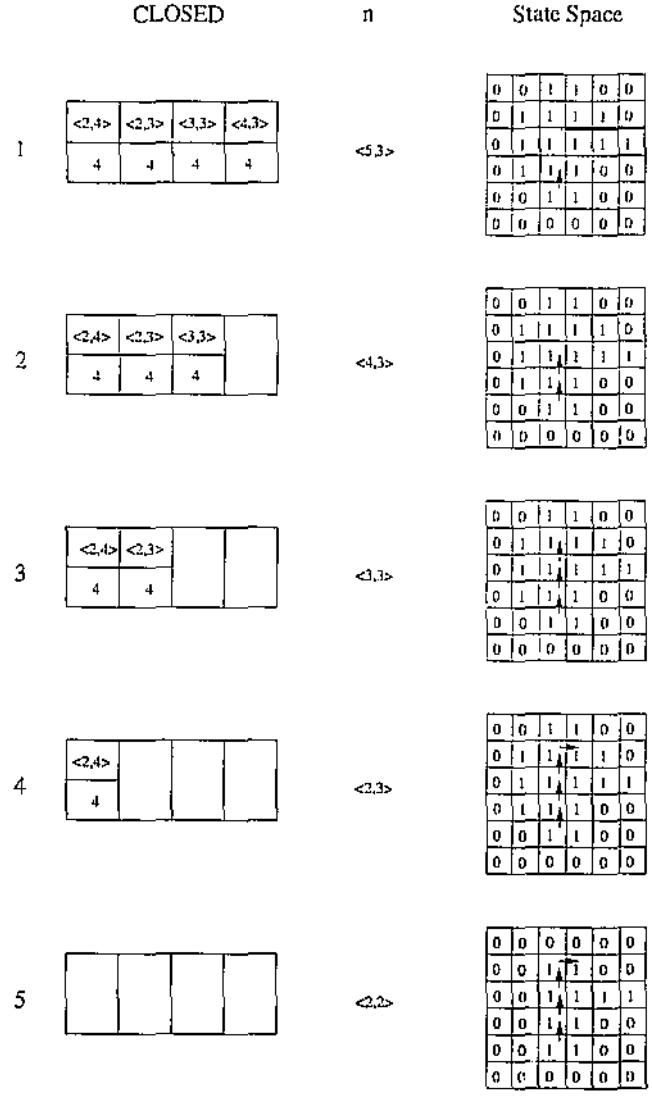


Figure 3.7: The process of backtrace and clearance

Proof. Let $q = |x_t - x_s| + |y_t - y_s| + |z_t - z_s|$. It can be seen that the length of the shortest path connection between s and t is equal to or greater than q, and that the length must be $q + 2j$, where $j = 0, 1, \cdots$. As mentioned earlier in this section, the exploration is a loop of 'selecting-and-expanding'. Once a selected state n has been expanded, new potential path connections are generated and kept in OPEN. In this way all the potential path connections between s and t are kept in OPEN at any time. Since this algorithm always select a state with minimum $f(n)$ to expand, the algorithm tries to find a path connection between s and t whose path length is q. If no such a path exists the algorithm then tries to find a path connection between s and t whose path length is $q + 2; \cdots$. This algorithm never explores the goal state t among the states which are $q + 2(i + 1)$ away from the initial state t before all the states which is $q + 2i$ away from the initial state t have been explored. Thus, this algorithm can guarantee to find the shortest path connection between s and t. □

The computational time of the algorithm consists of two parts: the computational time spent on the exploration and the computational time spent on the backtrace and clearance. First of all let us look at the computational complexity for the exploration. In the best case, that is where there is no obstacle between s and t, the search space just covers the states on the shortest path found and the neighboring states adjacent to those states on the path. Hence the total search space is limited to $5l$ and thus the computational complexity in this case is $O(l)$, where l is the length of the shortest path. In this case the search space for the Lee algorithm is $\frac{1}{3}l^3 + l^2 + \frac{7}{3}l$, that is $O(l^3)$ [104].

In the worst case, that is where there are a significant number of obstacles between s and t, the search space is bounded to the states on the paths between s and t whose length is less than or equals to l. Let $\Delta x = |x_t - x_s|$, $\Delta y = |y_t - y_s|$, and $\Delta z = |z_t - z_s|$; where $s =< x_s, y_s, z_s >$ and $t =< x_t, y_t, z_t >$. Then the total search space is bounded to a cubic container whose size is $(\Delta x + 2d) \times (\Delta y + 2d) \times (\Delta x + 2d)$, where $d = (l - (\Delta x + \Delta y + \Delta z))/2$. Figure 3.8 illustrates the search space.

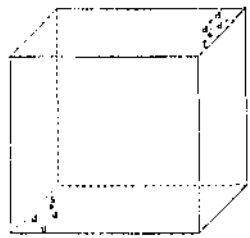


Figure 3.8: The search space in the worst case

Hence the search space $S = (\Delta x + 2d) \times (\Delta y + 2d) \times (\Delta z + 2d)$.

Since

$$l = (\Delta x + \Delta y + \Delta z) + 2d$$

Then

$$(\Delta x + 2d) + (\Delta y + 2d) + (\Delta z + 2d) = l + 4d.$$

When $(\Delta x + 2d) = (\Delta y + 2d) = (\Delta z + 2d) = \frac{l+4d}{3}$, the maximum search space $S_{Max} = (\frac{l+4d}{3})^3$. Therefore the computational complexity in this case is $O(l^3)$.

The computational time for the backtrace and clearance is proportional to the total number of states in OPEN and CLOSED, and thus proportional to the search space. Thus, the computational complexities for the backtrace and clearance are $O(l)$ in the best case and $O(l^3)$ in the worst cases, where l is the length of the shortest path. Hence the computational complexities for the A^* algorithm in the best case and in the worst case are $O(l)$ and $O(l^3)$ respectively.

The space used by this algorithm also consists of two parts: the space used for representing the state space M and the space used for OPEN and CLOSED. The space used for M is $m \times n \times p$ bits, while the space used for OPEN and CLOSED is proportional to the search space which is $O(l)$ in the best case and $O(l^3)$ in the worst case.

3.5 Case Study

Since there are no benchmarks available for the 3-D path connection problem, it is impossible for us to compare and contrast the performance of this algorithm with that of the other path connection algorithms. In order to demonstrate the performance of this proposed algorithm, we apply the algorithm on the two path connection instances shown in Figure 3.9 and compare and contrast the performance of this algorithm with that of the famous Lee algorithm. We would like to point out that the path connection problems shown in this figure are not given in grid graphs. In fact we present them in grid cell graphs for the convenience of representation. Since it is difficult to illustrate the search space for 3-D path connection problems, we use two 2-D path connection instances. Figure 3.9(a) shows a path connection problem in which there are no obstacles, while Figure 3.9(b) shows a path connection problem in which there are a significant number of obstacles.

Figure 3.10(a) and Figure 3.10(b) show the search spaces of our algorithm and the Lee algorithm, respectively, for the first path connection problem. Figure 3.11(a) and Figure 3.11(b) show the search spaces of our algorithm and the Lee algorithm, respectively, for the second path connection problem. For the first path connection our algorithm explored 31 grids while the Lee algorithm explored 221 grids. For the second path connection problem, our algorithm explored 62 grids, while the Lee algorithm explored 428 grid grids. It is not difficult to understand that the difference of search space between the two algorithms will become larger and larger as the size of problems and of the number of routing layers are increased.

3.6 Variations of the A^* Algorithm

In practical multi-layer VLSI routing, the shortest path connection is not just limited to finding the shortest path connection between two grid points. Sometimes we need to find the shortest path connection between two sub-wires. By sub-wire we mean a collection of interconnected wire segments. In addition, to minimize the

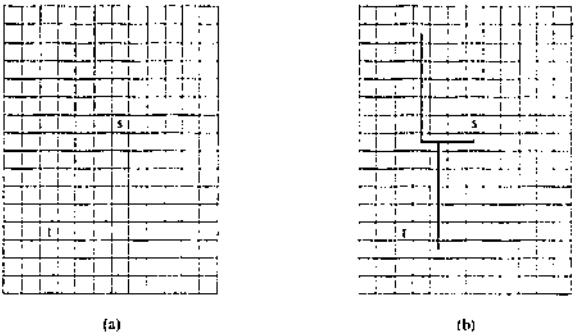


Figure 3.9: Two typical path connection problems

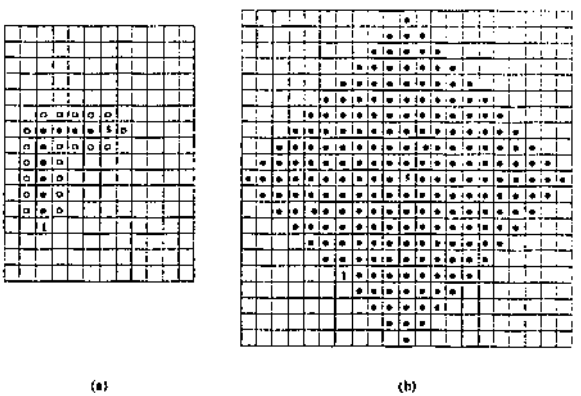


Figure 3.10: The search spaces for the first path connection problem

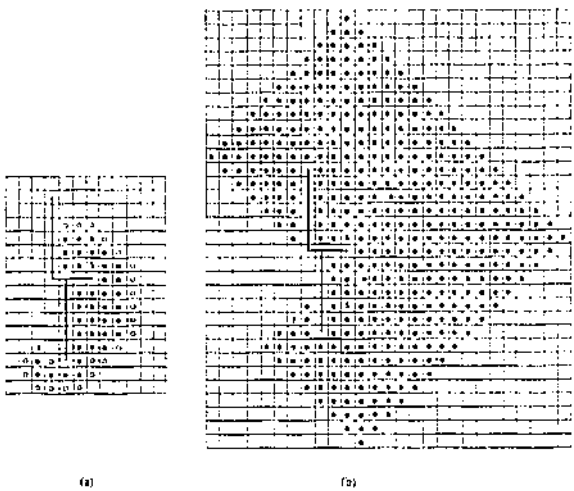


Figure 3.11: The search spaces for the second path connection problem

number of layer changes is desirable in multi-layer VLSI. Moreover, in performance-driven routing, *crosstalk* issue must be considered in a path connection algorithm. This section discusses three variations of A^* algorithm: the first one is the shortest path connection between two sub-wires; the second one is the shortest path connection with minimization of layer changes; and the third one is the shortest path connection with minimization of crosstalk.

3.6.1 The shortest path connection between two sub-wires

The shortest path connection problem between two sub-wires can be transformed into the state problem $P = (S, O, G)$, where S is the set of the states corresponding to the grid points on one sub-wire and G is the set of the states corresponding to the grid points on the other sub-wire. This path connection algorithm can be modified slightly to satisfy the new need. The following algorithm is the variation of the shortest path connection algorithm. In this variation, we combine exploration,

backtrace and clearance into one complete algorithm.

Algorithm 3.4 The Shortest Path Connection Between Two Subwires

1. *for* $\forall s \in S$, *push the state* s, *together with* $g(s)$ *and* $h(s)$, *onto OPEN;*
2. *if* $OPEN = \phi$ *then*
3. *exit with failure;*
4. *select a frontier state* n *from OPEN according to the heuristic information;*
5. *if* $n = t$ *then*
6. *go to 9;*
7. $M(n) = 1$;
8. *calculate* $f(n)$;
9. *remove* n *from OPEN and put it, together with* $f(n)$, *onto CLOSED;*
10. *expand* n *to find new frontier states and for each of the frontier states* q:
    (a) *calculate* $g(q)$;
    (b) *calculate* $h(q)$;
    (c) *push* q *together* $g(n)$ *and* $h(q)$ *onto OPEN.*
11. *go to 2;*
12. $q = t$;
13. *while* $q \notin S$ *do*
    (a) *find a neighboring state* p *of* q *in CLOSED such that* $f(p) = f(q)$ *or* $f(p) = f(q) - 2$;
    (b) *output* p;
    (c) *remove* p *from CLOSED;*
    (d) $q = p$;
14. *for* $\forall n \in OPEN or CLOSED$, $M(n) = 0$;
15. *exit with success.*

In fact, this variation of the path connection algorithm is used in the hybrid approach to switchbox routing which is presented in next chapter.

3.6.2 The shortest path connection with minimization of layer changes

To minimize the number of layer changes, we assign a penalty cost to traversing the grid in the direction of Z-axis. To implement this we simply replace Equation 3.2 with Equation 3.4.

$$g(p) = \begin{cases} 0 & \text{if } p = s \\ g(q) + 1 & \text{if } (p \neq s) \& (z = z_q) \\ q(q) + 2\lambda + 1 & \text{otherwise} \end{cases} \tag{3.4}$$

where q is the predecessor of the frontier grid point p, z and z_q are Z-axis coordinates of p and q respectively, $\lambda = m * n * l$.

In this way, it is guaranteed that the path connection found by this algorithm has minimal layer changes. It should be pointed out that the procedure of backtrace and clearance needs to be modified slightly. In this circumstances the predecessor state of n must be one of the six neighboring states stored in CLOSED and the value of $f(n)$ equals either $f(n)$, $f(n) - 2$, or $f(n) - 2\lambda - 2$.

3.6.3 The shortest path connection with minimization of crosstalk

In performance-driven VLSI routing, *crosstalk* is an important issue. Crosstalk is a parasitic coupling between neighbouring wire segments due to the mutual capacitance and inductance. Excessive crosstalk compromises noise margins, and possibly results in false receiver switching. Crosstalk can be minimized by making sure that no two wire segments are laid out in parallel or next to each other for

longer that a given parameter L_{max}. Our A^* algorithm can be modified slightly to meet this need.

What we have to do is to insert a field l_i into the node n_i in OPEN, where $i = 1, 2, \cdots$. The field l_i records the total wire length of the partly finished path which parallel or next to other wire segments. When l_i excesses L_{max}, we remove the node n_i from OPEN. In this manner we can guarantee that there is no path found by this A^* algorithm has no more than L_{max} length of wire segment that parallel or next to other wire segments.

3.7 Summary

In this chapter we proposed an A^* algorithm method to the 3-D shortest path connection problem for multi-layer VLSI routing. The new method transforms the shortest path connection problem into a state space problem of AI, and then uses A^* algorithm search technique to solve the state space problem. This A^* path connection algorithm has the following advantages:

- It can guarantee to find a path between a pair of terminals of a net if one exists and the path found is surely the shortest. The guarantee of finding an optimal solution is the major concern in VLSI routing. This is the key to the popularity of the Lee algorithm.
- It achieves high computational efficiency. This A^* algorithm method uses heuristic information to guide the search in the exploration phase and therefore reduces the search space. Since the search space is proportional to the computational time, this A^* algorithm method is computationally fast. It has been proven in this chapter that the computational complexity is $O(n)$ at the best case. Thus it is suitable to be used in a routing environment where the routed nets are spare.
- The requirement for the memory space is small. It can be seen from the

representation that it uses 1 bit for each grid point, while the other grid-based path connection methods uses at least 2 bit per grid. This economical representation benefits from our novel backtrace and clearance technique.

- It is flexible. This A^* algorithm method can be easily modified to meet different practical demands. The shortest path connection algorithm between two sub-wires, the shortest path connection with minimization of layer changes and the shortest path connection with minimization of crosstalk are all the evidences to support this claim.

It must be noted however that this algorithm needs two auxiliary data structures: OPEN and CLOSED. It can be seen that the total number of nodes in OPEN and CLOSED is the number of the explored grid points, or the search space. Therefore, when the search space is huge this algorithm needs a significant amount of memory space for OPEN and CLOSED. Thus we can conclude that this algorithm is suitable for those shortest path connection problems whose problem space is huge while its search space is relatively small. In other words, this algorithm is suitable to use in a large scale VLSI routing environment where the potential path length is not too long or there are not too many obstacles.

Chapter 4

Hybrid Approach to Switchbox Routing Problem

Switchbox routing is an intractable problem in VLSI design automation. From computational point of view, switchbox routing problem is NP-complete [42]. It is so difficult to tackle that no proposed approaches can guarantee to find a feasible routing solution even if one exists. In fact, no approach has been found to determine whether a switchbox routing problem is solvable or not.

Aiming at achieving high routing completion rate and high computational efficiency, we present a new approach to the switchbox routing problem in this chapter. This approach is a combined knowledge-based and algorithmic one, benefiting from both the high routing completion rate of knowledge-based approaches and the high computational efficiency of algorithmic approaches. Because of this, we also call this approach *hybrid approach*. Basically, this hybrid approach is based on a new knowledge-based routing technique, namely *synchronized routing*, we present in this chapter. It combines some modified well-known algorithmic routing techniques, such as *rip-up*.

In this chapter, we first formulate the switchbox routing problem. Then we outline this hybrid approach. After that we discuss the routing techniques used in the hybrid approach in detail, and present how to integrate the routing techniques into

a system. Next, the implementation issues are discussed and experimental results are given. Finally, we summarize this research.

4.1 Problem Formulation

A switchbox is formally defined as a rectangular region $R(h \times w)$, where h and w are positive integers. Each pair (i, j) in R is a grid point. The 0^{th} and h^{th} columns are the LEFT and RIGHT boundaries respectively. Similarly, the 0^{th} and w^{th} rows are the TOP and BOTTOM boundaries respectively. The connectivity and location of each terminal are represented as $LEFT[k] = n$, $RIGHT[k] = n$, $TOP[k] = n$, or $BOTTOM[k] = n$ depending upon the boundary of the switchbox it lies on, where k stands for the co-ordinate of the terminal alone the boundary and n is a net label specifying the net that the terminal belongs to. In particular, $n = 0$ means there is no terminal there.

Since it is assumed that the terminals are fixed on the boundaries, the routing area in a switchbox is fixed. Switchbox routing is to connect the terminals with the same label for all the nets on available routing layers and within the routing region subject to technological constraints. The goal of the switchbox routing is to interconnect all terminals belonging to the same net with minimum total length. Figure 4.1 is an example of two-layer switchbox problem and its solution. Here, $TOP = [0, 1, 5, 6, 4, 3, 0]$, $BOTTOM = [1, 2, 5, 2, 2, 0, 3]$, $LEFT = [1, 5, 0, 5]$ and $RIGHT = [4, 3, 6, 2]$.

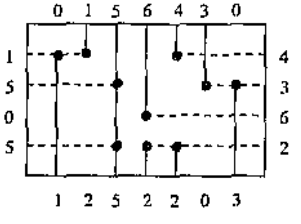


Figure 4.1: A switchbox routing problem and its solution

Notice that a two-layer switchbox routing problem is much more challenging than a multi-layer one because it has less resource (tracks) and thus has more conflicts amongst the nets in the switchbox. This research focuses on two-layer switchbox routing problem.

4.2 Outline of the Hybrid Approach

As discussed in Chapter 2, the existing approaches to the switchbox routing problem can be divided into two categories: algorithmic approaches and knowledge-based approaches. Generally speaking, algorithmic approaches are fast in their computation speed. In switchbox routing however, there are some aspects that do not lend themselves to an algorithmic approach. Modeling, for example, is one of the aspects. It is known that the crucial problem in solving the switchbox routing problem is how to find and resolve the conflicts among the nets in a switchbox. The conflicts result from the competition for the resource (tracks) in the switchbox. However, the conflicts are difficult or impossible to be modeled. Because of this reason all the algorithmic approaches are heuristic ones. As a result, when connecting a net, algorithmic approaches are not able to judge how the wires of one net will interact with that of the other nets, and thus they are not able to achieve high routing completion rate.

In contrast to algorithmic approaches, knowledge-based approaches can achieve remarkably high routing completion rate because they can use human routing experts' knowledge and expertise to discover and resolve potential conflicts among the nets in a switchbox. However, as the knowledge and expertise involved in the switchbox routing is extremely complex, the knowledge-based approaches have excessively large knowledge base(s). As a result, it is not easy to match the relevant knowledge in the large knowledge base(s) and resolve the conflicts among the matched knowledge. Due to this reason, the inference process on the knowledge base(s) is not efficient and consequently the computation speed is very slow. Another problem for knowledge-based approaches is that knowledge-based systems are usually

not standalone because they are implemented on a knowledge-based system development platform. Hence, they are difficult to be embedded into real VLSI routing systems. For example, WEAVER, the most successful knowledge-based system for switchbox routing, was implemented in OPS5.

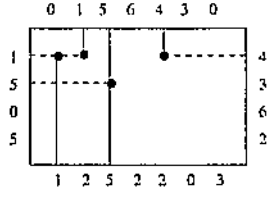
Aiming at achieving the high computational efficiency of algorithmic approaches and the competitive routing completion rate of knowledge-based approaches and overcoming their deficiencies, our hybrid approach makes use of five routing techniques: *pattern routing*, *corner routing*, *synchronized routing*, *rip-up* and *maze routing*. Some of these routing techniques are algorithmic ones while some of them are knowledge-based ones. Pattern routing is a geometry-based routing technique which is used for connecting some terminals that are in several special patterns. Corner routing is a knowledge-based routing technique which finds the connections at the four corners of a switchbox. Synchronized routing is another knowledge-based routing technique inspired by human experts' routing style. Rip-up dismantles some connections which prevent the other nets' connection. Maze routing is the variation of the 3-D shortest path connection algorithm which we presented in the previous chapter and used for connecting two sub-wires of a net whose connection involve a detour connection which synchronized routing is not compete to. Each of these routing techniques is used in a particular stage of switchbox routing. These routing techniques operate on the same design data and are harmonized by a central control program. They cooperate each other to achieve fast and quality switchbox routing.

4.3 Routing Techniques

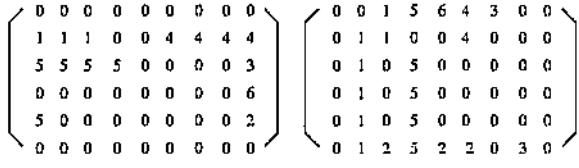
This section discusses in detail the routing techniques used in this hybrid approach. In order to facilitate the presentation we would like to introduce some terminologies and the intermediate status representation of a switchbox routing problem used throughout this chapter.

Row-tracks and *column-tracks* are the horizontal and vertical gird lines imposed in a switchbox respectively. *Sub-wire* is a maximal connected component of the wire segments of a net. *Active sub-wire* is the sub-wire that is being extended. *Extending point* is a properly selected point at a sub-wire from which the sub-wire is to be extended. *Active extending point* is the extending point on an active sub-wire.

The intermediate status of a two-layer switchbox routing problem R with h row-tracks and w column-tracks is represented in two $(h+2) \times (w+2)$ integer matrixes M_1 and M_2, which reflect the status on the first and the second layers respectively. If grid point $< i, j >$ on the first (second) layer is occupied by Net n, then $M_1[i,j] = n$ $(M_2[i,j] = n)$; otherwise $M_1[i,j] = 0$ $(M_2[i,j] = 0)$, where $0 \leq i \leq (h+1)$ and $0 \leq j \leq (w+1)$. Figure 4.2 is an example of the intermediate status representation. Figure 4.2(a) shows a partially completed solution to a switchbox routing problem and Figure 4.2(b) displays the status representation for the partially completed routing.



(a)



M_1 M_2

(b)

Figure 4.2: Problem representation

In order to indicate the positions of the extending points in a switchbox, we utilize a $(h+2) \times (w+2)$ binary matrix *Active*. $Active[i][j] = 1$ means that there is an extending point at the grid $< i, j >$ and $Active[i][j] = 0$ means that there is not any extending point there, where $0 \leq i \leq (h+1)$ and $0 \leq j \leq (w+1)$. Furthermore, we use $Row\_Min[i][n]$ and $Row\_Max[i][n]$ to represent the minimal and maximal number of Net n to be allowed to cross row-track i respectively, where $1 \leq i \leq h$. Initially they are computed by Equations 4.1 and 4.2:

$$Row\_Min[i][n] = min\{T[i][n], B[i][n]\} \tag{4.1}$$

$$Row\_Max[i][n] = max\{T[i][n], B[i][n]\} \tag{4.2}$$

where $T[i][n]$ and $B[i][n]$ are the number of the terminals at and above the row track i and the number of the terminals at and below the row track i respectively. Similarly, we denote $Column\_Min[j][n]$ and $Column\_Max[j][n]$ as the minimal and maximal number of Net n to be allowed to cross column-track j respectively, where $1 \leq j \leq w$. Initially they are computed by Equations 4.3 and 4.4:

$$Column\_Min[j][n] = min\{L[j][n], R[j][n]\} \tag{4.3}$$

$$Column\_Max[j][n] = max\{L[j][n], R[j][n]\} \tag{4.4}$$

where $L[j][n]$ and $R[j][n]$ are the number of the terminals at and above the column-track j and the number of the terminals at and below the column-track j respectively. The values of Row_Min, Row_Max, $Column\_Min$ and $Column\_Max$ are dynamically changed in response to the status of a switchbox during the process of the switchbox routing. Once a wire segment of Net n has passed cross the row-track i, for example, the values of $Row\_Min[i][n]$ and $Row\_Max[i][n]$ will be deducted by 1.

4.3.1 Pattern routing

In studying the successful routing solutions for the switchbox routing problems we found that although different switchbox routers produce different solutions for a switchbox routing problem, the connections for some particular terminal patterns are almost identical, and that for the same terminal pattern they always use the same way to connect it in solving different switchbox problems. Pattern routing is a special routing procedure which can identify some terminal patterns and connect them in specific manners.

There are two such terminal patterns. The first pattern is that two terminals of a net are at opposite parallel boundaries of a row or column. For this pattern a straight wire segment is used to connect them. This type of pattern routing is performed under the reserved routing model and serves as a priority routing in this hybrid approach. As a convention we assume that the terminals at the left and right boundaries originate from the same layer and the terminals at the top and bottom boundaries originate from the other layer. However, it will be shown in next chapter that a terminal can be arranged to exit on either layer in post-layout optimization. Algorithm 4.1 describes this.

Algorithm 4.1 The first type pattern routing

for $i = 1$ *to* h *do*
    if $Left[i] = Right[i]$ *then*
        $connect(< i, 0 >, < i, w+1 >, 1)$;
for $j = 1$ *to* w *do*
    if $Top[j] = Bottom[j]$ *then*
        $connect(< 0, j >, < h+1, j >, 2)$.

In this algorithm, the procedure $connect(< i_1, j_1 >, < i_2, j_2 >, l)$ produces a wire segment connecting grid points $< i_1, j_1 >$ and $< i_2, j_2 >$ on layer l, where $0 \leq$

$i_1, i_2 \leq (h+1)$, $0 \leq j_1, j_2 \leq (w+1)$ and $1 \leq l \leq 2$. Figure 4.3 shows the status of the Burstein's difficult switchbox [10] after applying this type of pattern routing. It can been seen from Figure 4.3 that this type of connection is the shortest connection for the pairs of terminals, and most importantly, it will not exert horizontal or vertical constraints on any other terminals and therefore the connection will not affect the succeeding connections for the other nets in the switchbox

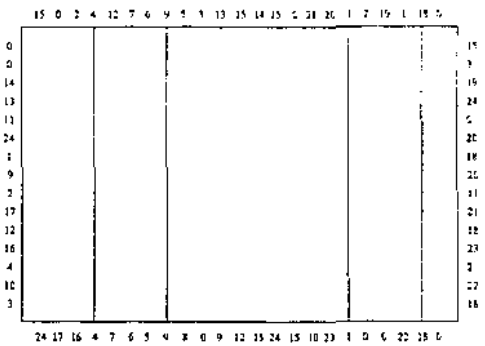


Figure 4.3: Illustration of the first type pattern routing

Another terminal pattern is that a pair of terminals of a net are on two adjacent boundaries. For this terminal pattern, we connect them in two perpendicular joint wire segments, one of which is on a layer and the other is on another layer and they are interconnected through a via at the joint grid point. Figure 4.4 illustrates this type of pattern routing.

In Figure 4.4, we assume that $Left[i] = Top[j] \neq 0$. We must point out that this pattern routing can be performed only when the connection will not affect the terminals at $Right[i]$ and $Bottom[j]$, if any. To determine if the connection will affect the terminal at $Bottom[j]$, we check if the terminal at $Bottom[j]$ is a terminal of the same net with the terminals at $Left[i]$ and $Top[j]$. If it is not, then we further check the value of $Row\_Min[i][Bottom[j]]$. If $Row\_Min[i][Bottom[j]] = 0$, then it means that a Manhattan connection for the net of the terminal at $Bottom[j]$ will be performed beneath row-track i, and therefore it will not be interacted by the

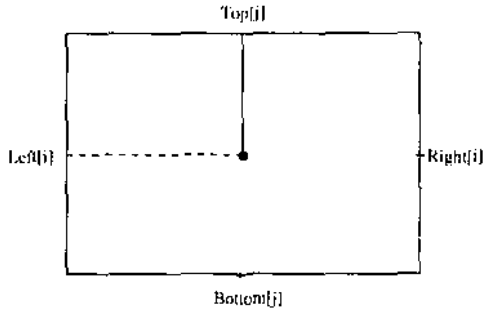


Figure 4.4: Illustration of the second type of pattern routing

net of the terminals $Top[j]$ and $Left[i]$. Similarly we can check if the connection will affect the terminal at $Right[i]$.

This pattern routing is implemented by scanning the switchbox, and generally it might need more than one pass to get it through. The reason for more than one pass scan is that a pair of terminals which cannot be connected in a pass might be connected in the following passes because of the instantly changing status of a switchbox routing. Figure 4.5 illustrates the process of this type of pattern routing based on the result of the first type pattern routing shown in Figure ??. This pattern routing is completed in two passes. Figure 4.5(a) shows the status of the switchbox after the first pass, and Figure 4.5(b) shows the status of the switchbox after the second pass. It can be seen from Figure 4.5(b) that a pair of terminals of Net 16 at Left[12] and Bottom[3] cannot be connected during the first pass because $Row\_Min[12][2] = 1$ at that time. However, after the two terminals of Net 2 at $Top[19]$ and $Right[13]$ are connected, $Row\_Min[12][2]$ becomes 0. Thus this pair of terminals are connected in the second pass.

Like the first type pattern routing, this pattern routing is also performed under the reserved routing model and used as a priority routing in this hybrid approach. The second pattern routing is presented in Algorithm 4.2.

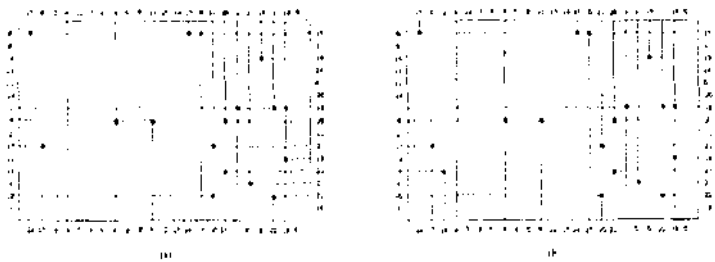


Figure 4.5: The process of the second pattern routing

Algorithm 4.2 The second type pattern routing

changed = *fasle*;
do
for $i = 1$ *to* h *do*
for $j = 1$ *to* w *do*
begin
if *Left*[i] = *Top*[j] *and they do not interact with* *Right*[i] *and* *Bottom*[j] *then*
begin
connect$(< i, 0 >, < i, j >, 1)$;
connect$(< 0, j >, < i, j >, 2)$;
Active[i][0] = 0;
Active[0][j] = 0;
modify relevant Row_Min and Column_Min;
changed = *true*
end
if *Left*[i] = *Bottom*[j] *and they do not interact with* *Right*[i] *and* *Top*[j] *then*
begin
connect$(< i, 0 >, < i, j >, 1)$;
connect$(< i, j >, < h + 1, j >, 2)$;
Active[i][0] = 0;
Active[h + 1][j] = 0;
modify relevant Row_Min and Column_Min;
changed = *true*
end

Algorithm 4.2 The second type of pattern routing (cont.)

if Right[i] = *Up*[j] *and they do not interact with Left*[i] *and Bottom*[j] *then*
begin
connect(< i, j >, < $i, w+1$ >, 1);
connect(< $0, j$ >, < i, j >, 2);
Active[i][$w+1$] = 0;
Active[0][j] = 0;
modify relevant Row_Min and Column_Min;
changed = *true*
end
if Right[i] = *Bottom*[j] *and they do not interact with Left*[i] *and Top*[j] *then*
begin
connect(< i, j >, < $i, w+1$ >, 1);
connect(< i, j >, < $h+1, j$ >, 2);
Active[i][$w+1$] = 0;
Active[$h+1$][j] = 0;
modify relevant Row_Min and Column_Min;
changed = *true*
end
end
until (*changed* = *false*).

In this algorithm, the logical variable '*changed*' monitors the change of the design data. Initially it is 'false'. When there is any change, it becomes 'true'. The variable is used in the termination condition of the algorithm. The procedure *connect*(< i_1, j_1 >, < i_2, j_2 >, l) produces a wire segment connecting grid points < i_1, j_1 > and < i_2, j_2 > on layer l, where $0 \leq i_1, i_2 \leq (h+1)$, $0 \leq j_1, j_2 \leq (w+1)$ and $1 \leq l \leq 2$.

4.3.2 Corner routing

It has been observed that the mutual constraints among the nets in the four corners of a switchbox are always the strictest in the switchbox. As the consequence of the mutual constraints there are only few options for the local connections at the four corners. Sometimes there is only one connection option in a corner. As a result, any other connections that violate the local connection pattern will not lead to a feasible routing. Corner routing finds the exclusive connections at the four corners of a switchbox shared by all feasible solutions. Corner routing is performed under the reserved routing model and also works as a priority routing. Figure 4.6 illustrates the basic idea behind corner routing.

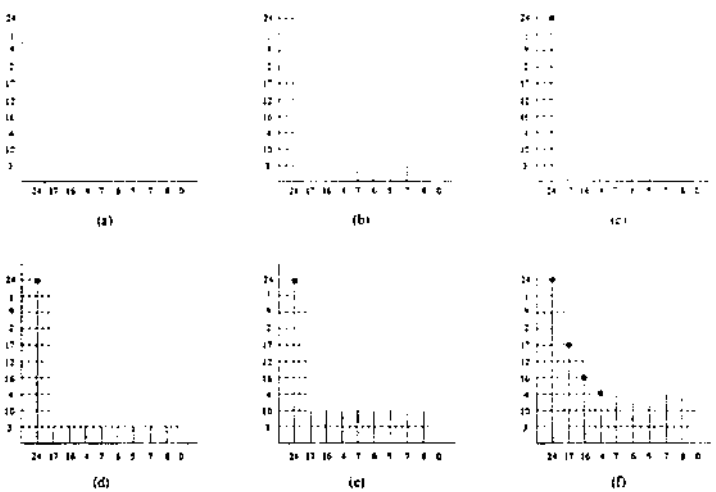


Figure 4.6: Illustration of corner routing

Figure 4.6(a) shows the initial status of the left-bottom corner of the Burstein's difficult switchbox routing problem. Initially, the terminals on the boundaries are extended into the switchbox by a unit length. Because corner routing is under reserved model, the terminals on the left boundary are extended on one routing

layer, say *horizontal layer* and the terminals on bottom boundaries are extended on another routing layer, say *vertical layer*. Figure 4.6(b) shows the status of the corner after the extension. It can been seen in this figure that a sub-wire of Net 3 and a sub-wire of Net 24 cross each other, and therefore the sub-wire of Net 24 can neither change layer at the grid point nor go to the right because there is a sub-wire of Net 17 in the next column track on the same layer. This forces the sub-wire to go up, and the extending point cannot stop until it meets another sub-wire of Net 24. Since the two sub-wires are on different layers, a via is introduced at the cross point to connect them. Figure 4.6(c) shows the status of the corner after extending the sub-wire of Net 24 at the bottom. After the extension, the sub-wires of the nets originated from the left boundary have to be extended to the right due to similar reasons. Figure 4.6(d) shows the status of the corner after extending those sub-wires. Similarly, those sub-wires originated from the bottom boundary have to be extended to the top as that is the only option for their extension. Figure 4.6(e) displays the status of the switchbox after extending the sub-wires. This process does not stop until there is no constraint imposed on any sub-wire at the corner of the switchbox. Figure 4.6(f) shows the status of the corner after the corner routing. In fact, that is the exclusive connections in the corner of the switchbox under reserved routing model. Algorithm 4.3 describes the corner routing.

Algorithm 4.3 Corner routing

```
for i=1 to h do
begin
    if Active[i][0] = 1 then
        extend(< i, 0 >, Left[i], 0);
    if Active[i][w + 1] = 1 then
        extend(< i, w + 1 >, Right[i], 0)
end
for j=1 to w do
begin
    if Active[0][j] = 1 then
    begin
        push the extending point (< 0, j >, Top[j], 1) onto a stack S;
        while (S ≠ φ) do
        begin
            pop the top element in S to P;
            extend(P);
            push the constrained extending points onto S
        end
    end
    if Active[h + 1][j] = 1 then
    begin
        push the extending point (< h + 1, j >, Top[j], 1) onto a stack S;
        while (S ≠ φ) do
        begin
            pop the top element in S to P;
            extend(P);
            push the constrained extending points onto S
        end
    end
end
```

In Algorithm 4.3, S is a stack to keep the information of extending points which are constrained in the context. The information covers the co-ordinates and the net label of the extending points and the layer they are located on. The procedure $extend(point, n, l)$ is used for extending a sub-wire of Net n on layer l from the

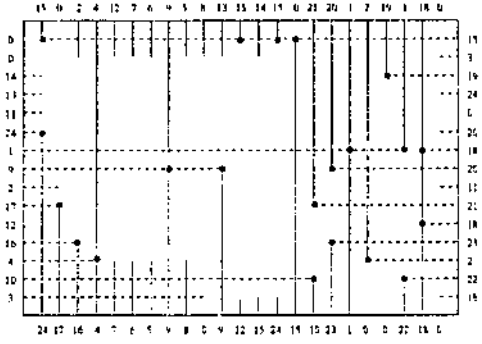


Figure 4.7: Corner routing result for Burstein's difficult switchbox problem

extending point *point*. This algorithm begins with extending the terminals (extending points) on the left and right boundaries into the switchbox. Because all the extensions are carried out on the horizontal layer, no constraint will be formed. Then, this algorithm extends the terminals (extending points) on the top and bottom boundaries one by one. Whenever any constraint is formed, this algorithm extends the constrained sub-wires immediately and never terminates until no more constrained sub-wires exist. Figure 4.7 shows the status of the Burstein's difficult switchbox after pattern routing and corner routing have been applied. Please note that corner routing also serves as a priority routing and comes after the pattern routing in this hybrid approach.

4.3.3 Synchronized routing

It has been observed that human routing experts are superior to switchbox routers in their high routing completion rate. What are the reasons for that? In which aspects human routing experts are different from conventional switchbox routers? In studying the human experts' routing process through experiment, we observed that:

- The human experts' switchbox routing process is exploratory or evolutionary in style. Initially, the designers only have a vague idea of the routing plan. As design proceeds, this plan is refined and frequently modified. The complete routing plan is not known until routing is completed and is usually not applicable, in detail, to another switchbox problem. This is in contrast to algorithmic approaches, which are usually built around a sequence of operations that is only very weakly influenced by the changing status and characteristics of the switchbox routing problems

- Human routing experts' routing process is in incremental style. They connect a net in several steps and consider multiple nets simultaneously according to the constraints among the nets which cannot be explicitly represented prior to the routing but can be found in the context

- Human routing experts' routing process is 'trial-and-error'. When a wire segment is blocked, they will backtrack or modify part of the routed wire segments

- Human experts have gained some amount of expertise and they can recognize and understand some special routing patterns and therefore use some special but effective techniques to cope with them

Inspired by the human experts' routing features, we develop a knowledge-based routing algorithm, namely *synchronized routing*. Synchronized routing is a 'multiple-net-at-a-time' routing technique. In other words, it takes into consideration multiple nets when extending a sub-wire of a net. The basic ideas behind synchronized routing are:

- A net's connection is completed by gradually extending the sub-wires of the net and the multiple nets among which there are mutual constraints grow synchronously in light of the mutual constraints among them in the context

- A sub-wire is extended by stretching a wire segment from an extending point on the sub-wire

- After a sub-wire has successfully been extended, synchronized routing examines if there are any sub-wires that are affected due to the extension. If there are some, we extend those sub-wires immediately. This recursive process does not terminate until no more sub-wire is constrained. In this way, those wire segments that have mutual constraints can been connected in synchronization

Synchronized routing is a recursion of 'selecting-and-extending'. 'Selecting' chooses a sub-wire and 'extending' expands the chosen sub-wire towards other sub-wire(s) of the net. Synchronized routing is depicted in Algorithm 4.4.

Algorithm 4.4 Synchronized routing

```
succ=false;
do
    select a sub-wire N among unconnected sub-wires;
    if there is no such a sub-wire then
        exit with success;
    select an extending point P on N;
    push P onto stack S;
    while (S ≠ φ) do
    begin
        pop the top node in S to Q;
        select an extending direction D for P;
        extend P and push the constrained extending points onto S;
        if the extending is successful then
            succ=true
    end
until (succ = false).
```

In Algorithm 4.4, the variable *succ* is used in the termination condition of the outer loop. If the variable *succ* becomes 'false' in the inner loop, that means that the extending point Q has been blocked. If this happens, we need to dismantle blocking wire segments by using rip-up which will be discussed shortly.

There are some issues involved in synchronized routing. The first issue is how to select a sub-wire. In this synchronized routing we select a sub-wire by scanning a switchbox from top to bottom and from left to right. Please note that this process is iterative. The second issue is how to determine an extending point for a sub-wire. This synchronized routing uses three heuristic rules to determine an extending point on a sub-wire:

1. If there is an end point on the sub-wire, then the end point is selected as an extending point; otherwise,
2. The point $< i, j >$ whose $Row\_Min[i][net] \div Column\_Min[j][net]$ is the maximum on the sub-wire is selected;
3. If there are more than one point whose $Row\_Min[i][net] \div Column\_Min[j][net]$ is the maximum, we give priority to the point at which a via is introduced if any.

The third issue is how to select an extending direction for the active extending point. The extending direction should lead to the other sub-wire(s) of the same net. To implement this, we can simply check all the extensible direction(s) and compare the values of $Row\_Min[i-1][net]$, $Column\_Min[j-1][net]$, $Row\_Min[i+1][net]$ and $Column\_Min[j+1][net]$, and choose the biggest one as its extending direction. For example, if $Row\_Min[i-1][net]$ is the biggest one, then the extending direction is toward the left boundary of the switchbox.

In order to better understand synchronized routing, let us look the process of synchronized routing for the Burstein's difficult switchbox shown in Figure 4.8. Figure 4.8(a) shows the status of the Burstein's switchbox after pattern routing and corner routing. Synchronized routing scans the extending points from the left to the right and from the top to the bottom and finds an extending point at grid point $< 2, 3 >$ as the active extending point. Then, synchronized routing chooses an extending direction for the prospective sub-wire in light of the heuristic rules, which is going down. After that synchronized routing extends the sub-wire,

since no any other net uses grid point $< 3,3 >$. The extension stops at that grid point. Figure 4.8(b) shows the status of the switchbox after the extending. Similarly, synchronized routing extends any sub-wire whose extending point is in the first row-track one by one from left to right. Figure 4.8(c) is the status of the switchbox after all the sub-wires in the first row-track have been extended. Then, the extending point at grid point $< 3,2 >$ is selected to extend. The sub-wire is initially extended to gird point $< 3,3 >$ and then the extending point at grid point $< 3,3 >$ is selected according to the selection rule for an active extending grid point and extended to grid point $< 3,12 >$. Figure 4.8(d) shows the status of the switchbox after the extension. As a result of the extension, those sub-wires whose extending point is on the newly extended wire segment are constrained and therefore they are extended immediately. Figure 4.8(e) shows the status of the switchbox after the constrained sub-wires have been extended. In this way, synchronized routing repeatedly selects the extending points from left to right and from top to bottom until no more sub-wire can be extended. Figure 4.8(f) shows the status of the switchbox after synchronized routing.

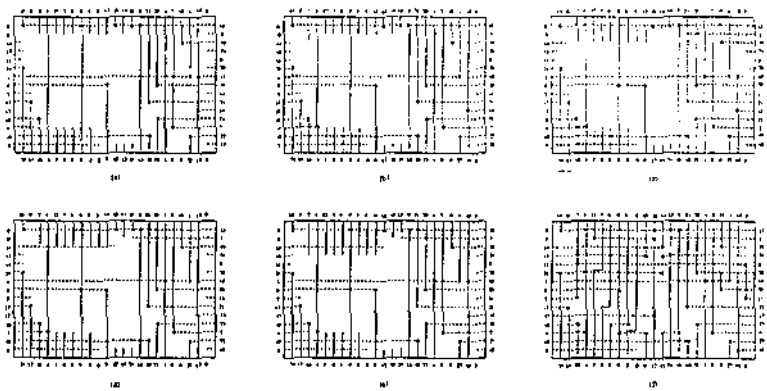


Figure 4.8: Illustration of synchronized routing

It is known that the routing completion rate is directly affected by the available space, or tracks, in a switchbox. Generally speaking, the bigger the available space,

the less interaction among the nets and therefore the higher possibility to achieve 100 percent routing completion rate. Because of this reason we use some techniques to evacuate space. There are three techniques used in this synchronized routing. The first technique is shown in Figure 4.9.

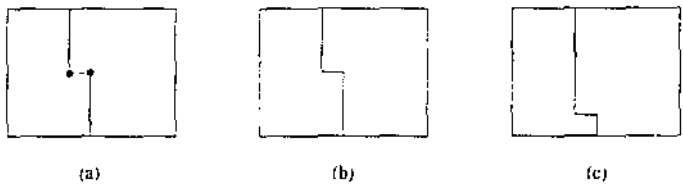


Figure 4.9: The first type of compact

Figure 4.9(a) shows the case before the compact. The compact is implemented in two steps:

1. Change the layer assignment of the horizontal wire segment from the broken layer to the solid layer (see Figure 4.9(b));
2. Shift the wire segment to either top boundary or bottom boundary depending on the situation (see Figure 4.9(c)).

Another type of compact is shown in Figure 4.10. This type of compact moves a wire segment in the middle of the switchbox towards a boundary. Figure 4.10(a) shows the case before the compact and Figure 4.10(b) illustrates the case after the compact.

It should be noted that this synchronized routing may produce some irrational connections. Figure 4.11(a) shows an instance of an irrational connection. In this case we need to erase the irrational connection part by the way shown in Figure 4.11(b). The elimination of irrational connections will also contribute to increasing the available space.

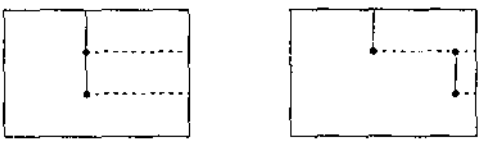


Figure 4.10: The second type of compact

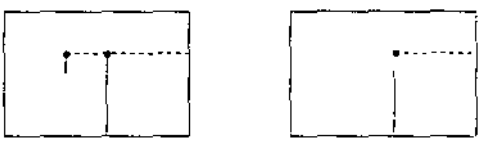


Figure 4.11: The third type of compact

4.3.4 Rip-up

In synchronized routing, a sub-wire might be blocked by other sub-wires and therefore prevents the sub-net from connecting with the other sub-wires of the net. If this happens, rip-up is employed to dismantle the blocking sub-wires such that the blocked sub-wire can be connected with the other sub-wires of the net.

Originally, the rip-up idea was proposed by Dees and Karger in [27]. Based on the observation that a large number of failures result from what is so-called 'isolation', that is when a sub-wire becomes isolated from the rest of the sub-wires, they proposed the so called rip-up and reroute technique which results in reasonably good connectivity improvement. The rip-up and reroute is a recursive process of three operations:

1. Rip-up the obstacle sub-wires;
2. Connect the blocked sub-wire;

3. Reroute the ripped sub-wires.

However, the rip-up used here is different from Dees and Karger's in the following two aspects:

1. This rip-up just removes the blocking part of the obstacle net rather than the whole obstacle net;

2. We divide the blocking nets into two categories: one is the net which had been completely interconnected before being dismantled, and the other is the net which had not been interconnected before being dismantled. If a dismantled net belongs to the first category, we reroute the dismantled net by using the maze routing technique which will be presented shortly; otherwise, we leave it to the next round of synchronized routing. Figure 4.12 illustrates the idea of rip-up.

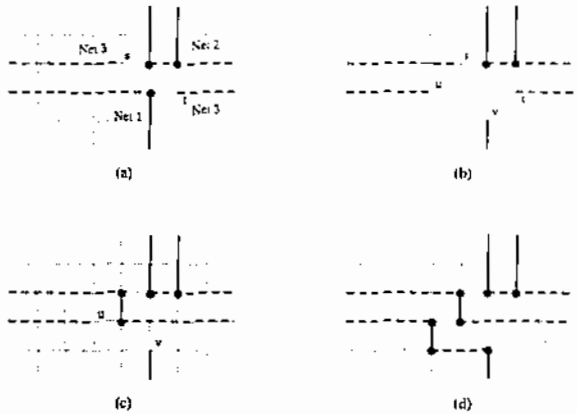


Figure 4.12: Rip-up

Figure 4.12(a) shows the status that two sub-wires of Net 3 are blocked by the routed sub-wires of Net 1 and Net 2. Rip-up identifies that Net 1 is the blocking

net and therefore dismantles the blocking part on Net 1. Figure 4.12(b) shows the status after part of Net 1 has been removed. After removing the obstacle part, the maze routing is invoked to interconnect the two sub-wires of Net 3. Figure 4.12(c) shows the status after the two sub-wires have been interconnected. Assume that Net 1 was a completed net, the maze routing is invoked again to interconnect the separated sub-wires of Net 1. Figure 4.12(d) show the status after the separated sub-wires have been interconnected.

4.3.5 Maze routing

The maze routing is the variation of the 3-D shortest path connection algorithm presented in the previous chapter. It seeks minimal length connection for two separated sub-wires.

4.4 Hybrid of Routing Techniques

In the last section we presented five different routing techniques, each of which is effective in solving a particular sub-problem and is used at a particular stage of switchbox routing. This section details how to integrate these routing techniques into a system, and demonstrates how they co-operate to solve switchbox routing problems.

4.4.1 System architecture

Figure 4.13 shows the architecture of our hybrid system. The figure shows that the hybrid system is composed of three parts: routers, common data area and central control program.

The routers are *pattern routing, corner routing, synchronized routing, rip-up* and *maze routing*, each of which corresponds to a routing technique described earlier.

The common data area is a collection of data structures which are accessible to these routers. The common data area comprises of two parts: *working memory* and *status*. The working memory includes M_1 and M_2, and the status contains *Active*, *Row_Min*, *Row_Max*, *Column_Min* and *Column_Max*. The initial status of the working memory is the switchbox to be routed and the eventual status of the working memory is the solution to the switchbox routing problem. Whenever a router is invoked and creates or dismantles a connection, the router modifies the contents of the working memory and the status in light of the change. In this way, the routers can communicate with each other. The central control program, which will be discussed in detail shortly, harmonizes the routers.

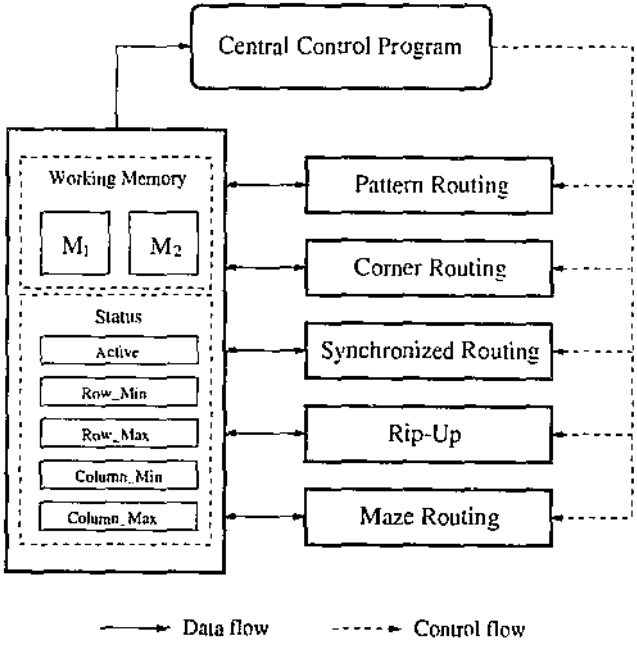


Figure 4.13: System architecture

4.4.2 Central control program

The central control program is a unit which harmonizes the routers. First of all, the program invokes pattern routing and then corner routing to produce a partially completed routing. This partially completed routing is fixed and never modified in the rest of routing. After that, synchronized routing is invoked to connect the unconnected sub-wires. If all the unconnected sub-wires are connected by synchronized routing, the routing is successfully completed. Generally speaking, synchronized routing has a major contribution to switchbox routing and can connect those unconnected sub-wires in most cases. However, it does not necessarily always work. Sometimes some sub-wires are blocked by other sub-wires. When this happens rip-up is invoked to identify and dismantle part of the blocking sub-wires. Then the maze routing is utilized to find the shortest path connection between the blocked sub-wire and any sub-wire which belongs to the same net with the blocked sub-wire. If the dismantled net was completely interconnected before being dismantled, the maze routing is invoked again to connect the separated sub-wires of the net. Then synchronized routing is invoked again if there still are unconnected sub-wires. Synchronized routing never terminates until either all the nets have been interconnected or the maze routing has failed. The former situation means that a solution to the switchbox routing problem has been found while the latter situation means that this hybrid system cannot find a solution for the switchbox problem. The flow chart of the central control program is shown in Figure 4.14.

In order to better understand this central control program, we illustrate the routing process of this hybrid routing system for the Burstein's difficult siwtchbox routing problem. The routing process is demonstrated in Figure 4.15. It can be seen from that figure that the whole routing is completed in five steps. First pattern routing is invoked. Figure 4.15(a) shows the status of the working memory after pattern routing. Next, corner routing is applied to find the exclusive connections at the four corners of the switchbox. Figure 4.15(b) shows the status of the working memory after corner routing. After that synchronized routing is used. Figure 4.15(c) depicts the status of the working memory after synchronized routing. It can be

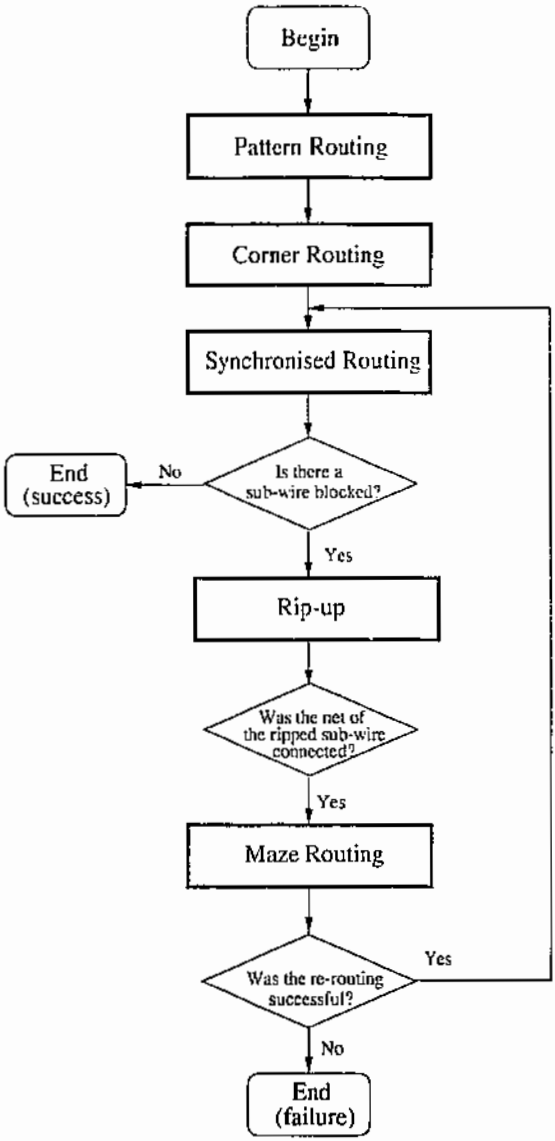


Figure 4.14: The flow chart of the central control program

seen that most of the nets in the switchbox have been connected by synchronized routing. The only net which has not been connected is Net 3. Net 3 has two separated sub-wires. The upper sub-wire is blocked by the connections of other nets. Thus rip-up is activated to identify and dismantle the blocking connections. Figure 4.15(d) describes the status of the working memory after rip-up. It can be seen from this figure that part of the wire segment of Net 24 is removed. The maze routing is then called to connect the two separated sub-wires of Net 3 and Net 24. Figure 4.15(e) is the final status of the working memory. At this point all the nets in the switchbox have been successfully connected.

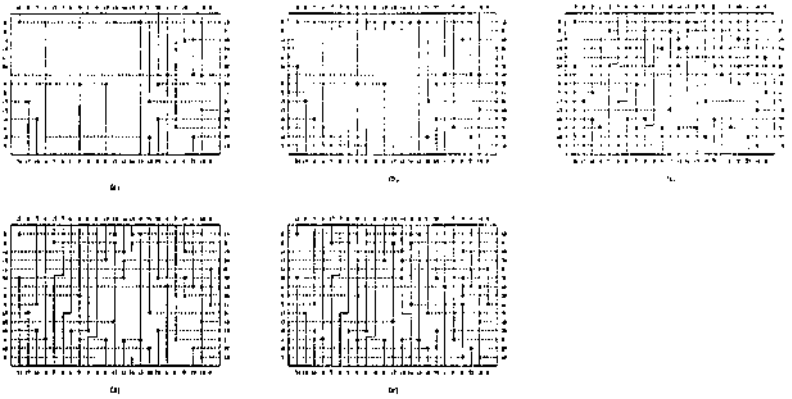


Figure 4.15: The routing process for the Burstein's difficult switchbox

4.5 Implementation and Experiments

The proposed hybrid approach has been implemented in C on a Pentium Pro 200 personal computer. The implemented switchbox routing system is called *HSR*. *HSR* contains 6,000 lines of source code.

As a convention we use HSR on the popular benchmarks for switchbox routing. The benchmarks include *difficult switchbox*, *more difficult switchbox*, *augmented*

dense switchbox, *modified dense switchbox*, *terminal intensive switchbox*, *sample switchbox* and *rectangle simple switchbox* [24]. For all of these benchmarks, HSR has successfully found a solution. Figure 4.16 to Figure 4.22 show the solutions to these benchmarks found by HSR.

As a comparison to other switchbox routers, we list the routing results obtained by different switchbox routers in Table 4.1 to Table 4.7. It can be seen from these results that HSR is comparable to the fastest algorithmic router BEAVER and its routing completion rate is as good as the best knowledge-based router WEAVER.

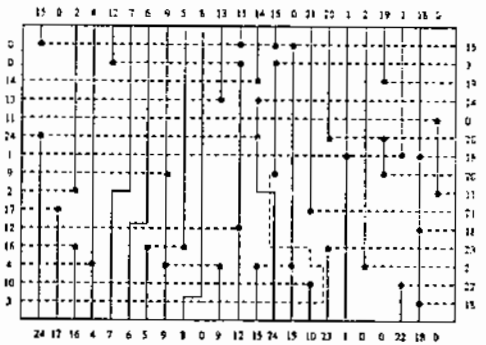


Figure 4.16: The solution to 'difficult' switchbox routing problem

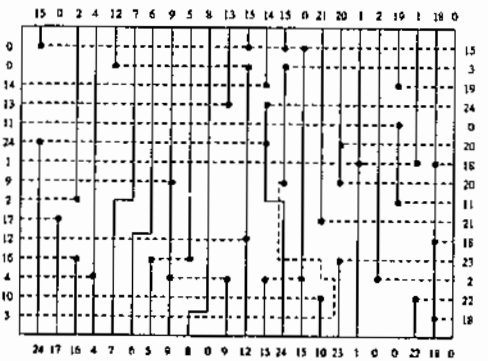


Figure 4.17: The solution to 'more difficult' switchbox routing problem

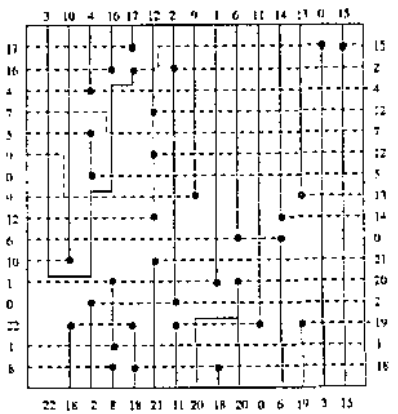


Figure 4.18: The solution to 'pedagogical' switchbox routing problem

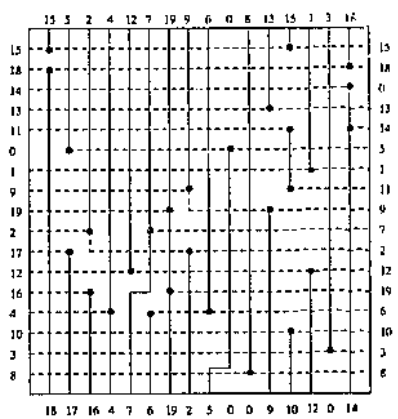


Figure 4.19: The solution to 'modified dense' switchbox routing problem

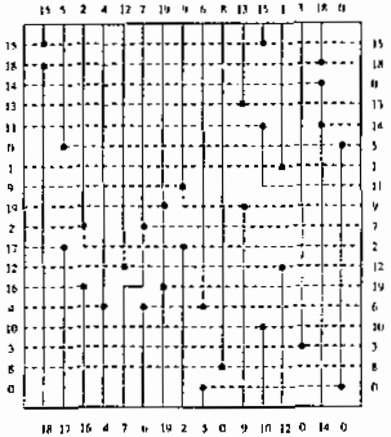


Figure 4.20: The solution to 'augmented dense' switchbox routing problem

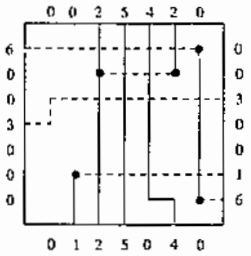


Figure 4.21: The solution to 'sample' switchbox routing problem

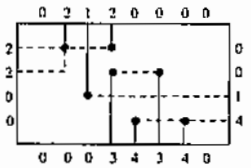


Figure 4.22: The solution to 'rectangle' switchbox routing problem

Router	*Vias*	*Length*	*Time(seconds)*
WEAVER	41	531	1508
BEAVER	35	547	1
MIGHTY	-	-	4
PACKER	45	546	56
CODAR	-	544	15
PARALLEX	-	-	25
HSR	41	538	1

Table 4.1: Router comparison for difficult switchbox

Router	*Vias*	*Length*	*Time(seconds)*
BEAVER	34	536	1
MIGHTY	39	541	4
PACKER	43	541	1400
CODAR	-	545	99
HSR	40	527	1

Table 4.2: Router comparison for more difficult switchbox

Router	*Vias*	*Length*	*Time(seconds)*
BEAVER	31	396	1
PACKER	45	406	91
HSR	34	425	1

Table 4.3: Router comparison for pedagogical switchbox

Router	*Vias*	*Length*	*Time(seconds)*
WEAVER	29	510	924
BEAVER	26	510	1
MIGHTY	29	510	-
PACKER	29	510	36
CODAR	-	510	31
PARALLEX	-	-	2
HSR	29	510	1

Table 4.4: Router comparison for modified dense switchbox

Router	*Vias*	*Length*	*Time(seconds)*
BEAVER	27	529	1
MIGHTY	32	530	4
PACKER	32	529	31
CODAR	-	529	10
HSR	31	529	1

Table 4.5: Router comparison for augmented dense switchbox

Router	*Vias*	*Length*	*Time(seconds)*
WEAVER	4	60	73
BEAVER	3	60	1
MIGHT	5	60	-
HSR	7	61	1

Table 4.6: Router comparison for sample switchbox

Router	*Vias*	*Length*	*Time(seconds)*
BEAVER	1	31	1
HSR	7	32	1

Table 4.7: Router comparison for simple rectangle switchbox

4.6 Summary

In this chapter a hybrid approach to switchbox routing problem was presented. The motive was to achieve the high routing completion rate of the knowledge-based approaches and the high efficiency of the algorithmic approaches by combining effective knowledge-based routing techniques and efficient algorithmic routing techniques.

This chapter detailed the routing techniques used in the hybrid approach and discussed in detail the integration of the routing techniques. It can be seen from the experimental results that this new approach can achieve competitive high routing completion rate. In fact, it successfully completed the routing for almost all of the popular switchbox routing problems. It can also be seen from the experimental results that the computation efficiency of our approach is remarkably high. For each of the popular switchbox problems, it took no more than 1 second to find a solution. Thus we can conclude that the routing completion rate of this approach is comparable to, if it is not better than, that of the previously best known switchbox routers, and the computational efficiency is comparable to that of the previously best known switchbox router.

This approach has a very high degree of parallelism and thus it can be developed into a parallel switchbox router to further improve the efficiency of routing. In addition, although synchronized routing is a kind of knowledge-based routing technique, it does not need any knowledge base because the knowledge is created in the context. As a result, we do not need any knowledge-based system development tool, and thus it can be embedded into a real VLSI routing system.

However, this approach only considered high routing completion rate and minimal total wire length as its only metrics. Hence some metrics, such as number of vias, were ignored during the switchbox routing. Thus, the routing solutions must be optimized before actually being put into use. In the following chapter we will discuss how to minimize the number of vias.

Chapter 5

Genetic Algorithm Approach to Constrained Via Minimization Problem

Via is a mechanism (hole) for connecting wire segments of a net distributed on different layers in two-layer or multi-layer VLSI routing. However, via has an associated resistance that affects circuit performance. In addition, in integrated circuit fabrication, the yield is inversely related to the number of vias because a chip with more vias has a smaller probability of being fabricated correctly. Besides, the size of a via is usually larger than the width of wires. As a result, more vias lead to more routing space. Therefore, it is desirable to minimize the number of vias introduced in VLSI routing. However, VLSI routing is a complex and intractable problem [113]. Therefore, existing routers and design tools can only consider the minimization of the number of tracks in channel routing, completion of switchbox routing, and wire length minimization as their primary objectives. As a consequence, via minimization is either ignored or de-emphasized in the routers and design tools, and therefore comes as a post-layout optimization problem in VLSI routing.

From the computational point of view, the constrained via minimization is NP-complete [72, 1]. For a special case where the number of the wire segments split at a via candidate is no more than three, some elegant theoretical results have been obtained [80, 18, 58, 5]. In practical VLSI routing, however, the situation that four or more wire segments split at a via candidate cannot be avoided. As a result, those methods are not suitable for handling practical constrained via minimization problems. Although several methods can be used to cope with the routing in which there are more than three wire segments split at a via candidate, they are only suitable for some particular routing styles. For example, Xiong and Kuh's method [117] can only be used for Manhattan or knock-knee routing. Another problem for the other existing methods is that they are difficult to be adjusted to meet some practical requirements, such as the layer assignment for some special nets like power and ground nets.

This chapter presents the first genetic algorithm approach to the constrained via minimization problem. This new approach can be used for both grid-based and gridless routing, and arbitrary number of wire segments are permitted to be split at a via candidate. This chapter is organized as follows: first, we formulate the constrained via minimization problem. Then, a new graph-theoretic model for the constrained via minimization problem is proposed. After that, on the basis of the new graph-theoretic model, we present the genetic algorithm in detail. This is followed by implementation and experiments. Finally, we summarize the genetic algorithm approach.

5.1 Problem Formulation

Given an initial routing consisting of a set of wire segments, the problem to reassign wire segments to available layers so that the logical connections are maintained and the number of vias required is minimized without changing the topology of the initial routing. Since the constrained via minimization is realized by assigning net segments to layers, the problem is also called the *layer assignment problem*.

Figure 5.1 shows an instance of the constrained via minimization problem, i.e. Figure 5.1(a) is an initial routing for a switchbox and Figure 5.1(b) gives the routing after the constrained via minimization. In this figure, dotted lines represent the wire segments on one routing layer, solid lines stand for the wire segments on the other routing layer and dots are vias. It can be seen that there are six vias introduced in the initial routing and there is only one via after the constrained via minimization.

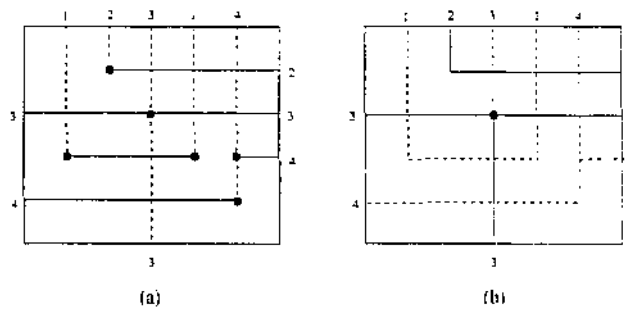


Figure 5.1: An instance of the constrained via minimization problem

5.2 Switching Graph Model

5.2.1 Terminology

In order to facilitate the description of the new graph-theoretic model, we first introduce some terminologies used in the rest of this chapter.

Via candidate: An appropriate position at which a via may be introduced. In Figure 5.2, for example, the positions v_1, v_2, and v_3 are via candidates.

Net segment: A piece of net separated by via candidates. For example, the line segments marked with n_1, n_2, $\cdots$, n_8 in Figure 5.2 are net segments. Note that a net segment does not have to be a straight line segment. It can be a collection of

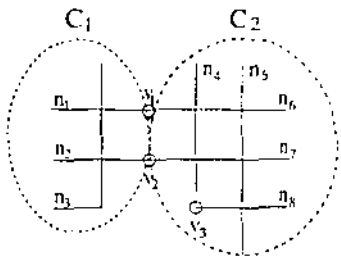


Figure 5.2: Illustration of terminologies

interconnected line segments. Net n_3, for instance, is such a net segment.

Cluster: A maximal set of mutually crossing or overlapping net segments. The net segments in Figure 5.2 constitute two clusters c_1 and c_2. Note that once a net segment is assigned to a layer, the layer assignments for the rest of the net segments in the cluster are determined. Because the net segment can be assigned onto either of the two available routing layers, there are only two possible layer assignment options for each cluster. The layer assignment for a net segment in a cluster only affects the layer assignments for the net segments in the cluster and never affects the layer assignment for a net segment in the other clusters.

Internal via candidate and *External via candidate*: An internal via candidate is referred to as a via candidate whose adjacent wire segments belong to the same cluster, while an external via candidate is referred to as a via candidate whose adjacent wire segments belong to more than one cluster. For example, in Figure 5.2 v_3 is an internal via candidate while v_1 and v_2 are external candidates.

Internal via and *External via*: An internal via is a via introduced at an internal via candidate, and an external via is a via introduced at an external via candidate.

Vias introduced at internal via candidates cannot be eliminated in any constrained via minimization technique because once the layer assignment of a wire segment changes the rest wire segments in the cluster need to be changed in order to keep

the feasibility (otherwise it will lead to short-circuit). As a result, the relative layer assignment for those net segments never change. Hence, if there is a via introduced at the internal via candidate, then there will still be a via there after constrained via minimization; if there is no via introduced at the internal via candidate, then there will be no via introduced there after constrained via minimization.

Feasible layer assignment: A feasible layer assignment is a layer assignment in which no two net segments of different nets are interconnected and all the net segments of the same net are interconnected.

5.2.2 Determination of via candidates, net segments and clusters

In determination of via candidates, net segments and clusters we must guarantee that the layer assignments for the net segments cover all the possible layer assignments of the initial routing. From this point of view, the net segments should be as fine as possible. However, if the net segments are too fine, it will lead to the increase of the size of problem representation. Thus, the net segments should be as large as possible as long as they can cover all the possible layer assignments of the initial routing. These are the basic ideas behind the determination of via candidates, net segments and clusters.

We must point out that the via candidates are not restricted to the positions at which vias are introduced in the initial routing. Often, in order to minimize the total number of vias a via might be introduced at a position where there is no via in the initial routing. In addition, the net segments used in the layer assignment process may not accord to the net segments in the initial routing. Hence, the first step for the constrained via minimization problem is to determine via candidates and net segments such that all the possible layer assignments are covered.

Given an initial routing, whether it is a grid-based routing or gridless routing, we can manage to produce its *planar representation* [116]. Planar representation is a

projection of the initial routing onto a plane that parallels to the routing layers of the initial routing. Figure 5.3 is the planar representation for the routing shown in Figure 5.1(a). From the planar representation we can determine *cross points* [116]. Cross points are those positions at which different nets intersect or overlap. The positions marked with cross symbols in Figure 5.3 are cross points. Via candidates then can be decided by means of the following two rules: firstly, a cross point cannot be chosen as a via candidate; secondly, if there are several contiguous positions on a net that are not cross points, we only choose one of them as a via candidate rather than all of them. In fact, we always choose the one with maximal split degree as a via candidate. Using the above two rules we can determine all the via candidates from the planar representation for the routing shown in Figure 5.1(a). These via candidates are marked with circle symbols and labeled with v_1, v_2, ..., v_7 in Figure 5.3. Once via candidates have been determined, net segments are obtained automatically. As shown in Figure 5.3, n_1, n_2, ..., n_{14} are all the net segments for this layer assignment. Hence, we find entire clusters as below:

$c_1 = \{n_1\}$;

$c_2 = \{n_2, n_8\}$;

$c_3 = \{n_3, n_{10}, n_{12}\}$;

$c_4 = \{n_4, n_6, n_7, n_9, n_{14}\}$;

$c_5 = \{n_5\}$;

$c_6 = \{n_{11}\}$;

$c_7 = \{n_{13}\}$.

5.2.3 Generation of a feasible layer assignment

Given all clusters, a feasible layer assignment can be obtained by forcing anyone of the net segments in each cluster to one of the two routing layers. Figure 5.4 shows

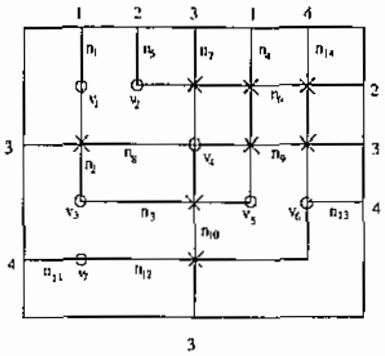


Figure 5.3: Planar representation

a feasible layer assignment for the new net segments. It is obtained by assigning n_1 in c_1, n_2 in c_2, n_4 in c_4, n_5 in c_5, n_{11} in c_6 onto dotted routing layer and assigning n_3 in c_3 and n_{13} in c_7 onto solid routing layer.

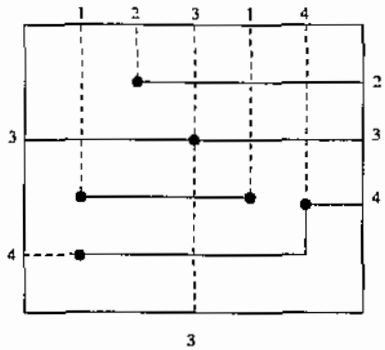


Figure 5.4: A feasible layer assignment

5.2.4 LAP graph

Assume that $c_1, c_2, \ldots, c_n$ are entire clusters, and $v_1, v_2, \ldots, v_p$ are all via candidates. Then a feasible layer assignment R can be represented in a layer assignment graph, or LAP graph for short, $G(V, E)$, which is constructed in the following way:

Let $V_{cluster} = \{c_1, c_2, \ldots, c_n\}$, $V_{via} = \{v_1, v_2, \ldots, v_p\}$, and $V = V_{cluster} \cup V_{via}$. For $\forall < u_1, u_2 > \in E$ if and only if either of the following conditions is satisfied:

1. $u_1 \in V_{via}$, $u_2 \in V_{cluster}$, and via candidate u_1 is adjacent to a net segment in the cluster u_2 and the net segment is assigned to the solid line layer in R;
2. $u_1 \in V_{cluster}$, $u_2 \in V_{via}$, and via candidate u_2 is adjacent to a net segment in the cluster u_1 and the net segment is assigned to the dotted line layer in R.

Figure 5.5 is the LAP graph corresponding to the feasible layer assignment shown in Figure 5.4.

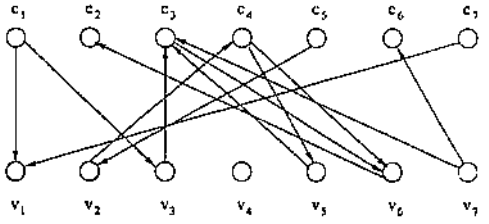


Figure 5.5: LAP graph

LAP graph is a *bigraph* [28]. In a LAP graph, the vertices belonging to $V_{cluster}$ are *cluster vertices* and the vertices belonging to V_{via} are *via candidate vertices.* The LAP graph reflects the mutual constraints among the via candidates and clusters, and possesses the following useful properties:

Property 5.1. The number of the via candidate vertices whose in-degree and out-degree are not zero in a LAP graph equals to the number of the vias introduced in the corresponding layer assignment R.

Proof. If both the in-degree and out-degree of a via candidate vertex are not zero, then at the via candidate in the corresponding layer assignment there must be some wire segments adjacent to the via candidate arranged on a layer while there are some other wire segments are on another layer according to the construction of LAP graph. Therefore, in order to connect them a via must be introduced at the via candidate. On the other hand, if both the in-degree and the out-degree of a via candidate vertex are zero, then at the via candidate in the corresponding layer assignment all the wire segments adjacent to the via candidate are arranged on the same layer. Thus, there is no need to introduce a via at the via candidate. Hence, the number of the via candidate vertices whose in-degree and out-degree are not zero in a LAP graph equals to the number of the vias introduced in the corresponding layer assignment. □

In the rest of this chapter, we call the vertices whose both in-degree and out-degree are not zero *via vertices* and the vertices whose in-degree or out-degree is zero *non-via vertices*. Denote the in-degree and out-degree of a via candidate vertex v as $id(v)$ and $od(v)$ respectively. Then we have Property 5.2 as described below:

Property 5.2. For $\forall v \in V_{via}$, $id(v) + od(v) \leq 4$.

Proof: It can be seen from the planar representation that a via candidate is adjacent to at most 4 net segments. Reflecting on the corresponding LAP graph, it means that there are at most four cluster vertices adjacent to the corresponding via candidate vertex, that is,

$$id(v) + od(v) \leq 4. \quad \square$$

5.2.5 Switching graph problem

We define a *switching graph* $S_G(C)$ of a LAP graph $G = (V, E)$ as the bigraph which is obtained by reversing the direction of the arcs incident to the cluster vertices in the cluster subset C. $S_G(C) = (V_G, E_G)$ is formally defined as below:

$$V_G = V;$$
$$E_G = E - \{< u_1, u_2 > | (< u_1, u_2 > \in E).and.((u_1 \in C).or.(u_2 \in C))\}$$
$$+ \{< u_2, u_1 > | (< u_1, u_2 > \in E).and.((u_1 \in C).or.(u_2 \in C))\}.$$

For example, the switching graph $S_G(\{c_3, c_6\})$ of the LAP graph shown in Figure 5.5 is displayed in Figure 5.6.

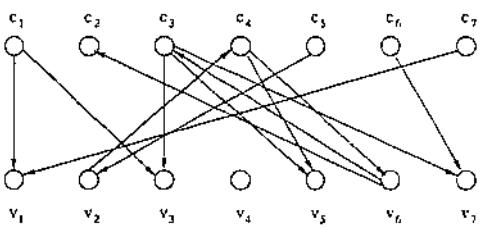


Figure 5.6: Switching graph

Theorem 5.1. Assume that N is the set of net segments, L is the set of all feasible layer assignments of N, l is a feasible layer assignment of N, G is the LAP graph of l, and S is the set of the switching graphs of G. Then, S and L are one-to-one correspondence.

Proof. First of all, we prove that a switching graph $S_G(C)$ corresponds to a feasible layer assignment. By a feasible layer assignment we mean that layer assignment in which all the wire segments of a net are interconnected and no net segments which belong to different nets are connected.

From the definition of switching graph we know that a switching graph corresponds to a layer assignment in which the layer assignment for those net segments in the

clusters of C has changed in relation to the layer assignment corresponding to the LAP graph G. From the definition of cluster we know that changing the layer assignment for all the net segments in a cluster will not affect the connectivity of the net segments in the cluster. Thus, the new layer assignment is still feasible. Therefore, a switching graph corresponds to a feasible layer assignment.

On the other hand, given a feasible layer assignment of N, we can construct a LAP graph G'. Suppose that C is the set of clusters in which the layer assignment of a cluster is different from the layer assignment corresponding to G. Then it is not difficult to see that $G' = S_G(C)$. In other words, a feasible layer assignment corresponds to a switching graph of G. □

Corollary 5.1. The layer assignment corresponding to $S_G(C)$ is a feasible one, where $C \subseteq V_{cluster}$.

For instance, the layer assignment corresponding to the switching graph $S_G(\{c_3, c_6\})$ shown in Figure 5.6 is displayed in Figure 5.7.

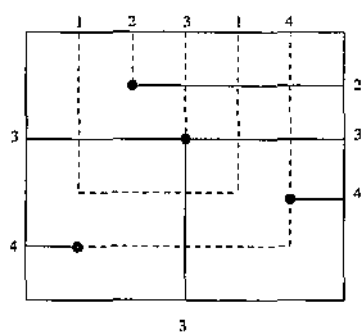


Figure 5.7: The feasible layer assignment corresponding to $S_G(\{c_3, c_6\})$

Corollary 5.2. There are 2^n different feasible layer assignments for R.

Proof. The total number of the feasible layer assignments is: $C_n^0 + C_n^1 + \cdots + C_n^n$, that is, 2^n. □

Corollary 5.3. The layer assignment with minimal number of vias corresponds to the switching graph with minimal number of via vertices.

So far, we have transformed the constrained via minimization problem into the switching graph problem problem described as: *given a LAP graph G, find the switching graph graph with minimal number of via vertices.* Therefore, given a routing R_0, the constrained via minimization problem can be solved through following steps:

1. Build the planar representation of R_0, P;
2. Determine via candidates V_{via}, and net segments N based on P;
3. Produce clusters $V_{cluster}$;
4. Generate a feasible layer assignment l;
5. Construct the LAP graph of l, G;
6. Find the switching graph of G such that it has minimal number of via vertices, S_{min};
7. Output the layer assignment corresponding to S_{min}.

5.2.6 Practical considerations

Due to the fabrication technology or performance considerations, some practical constraints should be taken into account within the constrained via minimization. Constraints are different from one routing environment to another subject to fabrication technology, design requirements and performance considerations. The switching graph model can be easily adjusted to meet different practical constraints. Here are some examples.

5.2.6.1 Special nets

In practical routing, some special nets are required to be assigned onto a particular layer. For example, power and ground nets are always assigned to the thicker metal layers. This switching graph model can meet the need by configuring the LAP graph.

First, we identify cluster vertices associated to the special nets. Next, we adjust the layer assignment for the clusters so that the special nets are assigned on the desired layer on the LAP graph. Then, we combine the cluster vertices to form a 'super' cluster vertex. The following example demonstrates this technique.

Suppose Net 2 in the constrained via minimization problem shown in Figure 5.1(a) is such a special net and the net segments of Net 2 are required to be arranged on the same layer, say, a more thicker metal layer. Net 2 has two net segments n_5 and n_6, which belong to cluster c_5 and c_4 respectively. Because in the feasible layer assignment (see Figure 5.4) corresponding to the LAP graph (see Figure 5.5) the net segments n_5 and n_6 are on different layers, we alter the layer assignment for the cluster c_4 such that net segments n_5 and n_6 are arranged on the same layer, and combine clusters c_5 and c_4 to form a super cluster $c_{4,5}$. The configured LAP graph is shown in Figure 5.8.

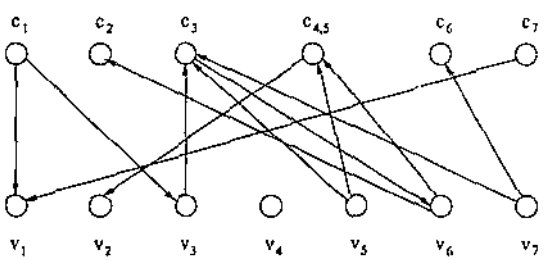


Figure 5.8: LAP graph

5.2.6.2 Fixed layer terminals

It is very common to assume that all terminals are available on both layers in theoretical discussions of routing and via minimization. However, in some practical cases terminals are only available on one layer. Sometimes, some of the terminals are forced to be arranged on a layer but the rest on are arranged the other layer. In such cases a significant number of terminals have to be vias in order to satisfy the assumption of the router.

This problem can be resolved by using the similar way we used in handling special nets. First, we identify cluster vertices associated to the terminals. Next, we adjust the layer assignment for the clusters so that the terminals are assigned on the desired layer on the LAP graph. Then, we combine the cluster vertices to form a 'super' cluster vertex.

5.3 Overview of Genetic Algorithm

Genetic algorithm is a search algorithm modeled on the mechanics of natural selection and natural genetics [50]. It belongs to the class of probabilistic algorithms, yet it is different from random algorithms as it combines elements of directed and stochastic search methods. As a result, genetic algorithm can efficiently exploit history information to speculate on new search points with expected improved performance. Due to this mechanism, genetic algorithm is more robust than heuristic algorithms and hence has been widely used for solving complex, large-scale and intractable problems in many areas [36, 15].

In genetic algorithms, domain-specific knowledge is embedded in the abstract representation of a candidate solution termed an *individual*. Individuals are grouped into a set called *population*. Successive populations are called *generations*. To begin with, a genetic algorithm creates an initial generation, $G(0)$, and for each generation, $G(t)$, generates a new one, $G(t+1)$. An abstract view of genetic algorithms

is given below:

Algorithm 5.1 Genetic Algorithm

create initial population $G(0)$;
evaluate $G(0)$;
$i = 1$;
repeat
  $t = t + 1$;
  generate $G(t)$ *using* $G(t-1)$;
  evaluate $G(t)$;
until solution is found.

In the above algorithm, the operation 'evaluate $G(t)$' refers to the assignment of a figure of merit to each of the population's individuals. Often an individual contains a single *chromosome*. A chromosome of length n is a vector of the form

$$< x_1, x_2, \cdots, x_n >,$$

where each x_i is an *allele*, or *gene*. The domain of values from which x_i is chosen is called the alphabet of the problem. Frequently, the alphabet used consists of the binary digits $\{0, 1\}$. A *schema* is a pattern representing a subset of chromosomes with similarities. For example, '$**11*0$' is a schema over the binary alphabet. The char '$*$' in the schema stands for "don't care". Therefore, the schema represents the chromosomes whose the third and fourth genes are '1' and whose the sixth gene is '0'. The *order* of a schema H, denoted by $o(H)$, is the number of fixed positions of the pattern. The *defining length* of a schema H, denoted by $\delta(H)$, is the distance between the first and the last specific genes. For example, $o(**11*0) = 3$, and $\delta(**11*0) = 3$. The fundamental theorem of genetic algorithms is schema theorem [50]:

Short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations of a genetic algorithm.

5.4 Outline of the Genetic Algorithm for Switching Graph Problem

This genetic algorithm is a knowledge-based one. It operates on a population of individuals, each of which represents a switching graph. Initial population is created by generating a collection of individuals. Each individual i is evaluated using a fitness function $f(i)$. The genetic algorithm discovers better individuals by allowing the individuals to evolve from generation to generation, and the evolution is realized through reproduction.

The process of reproduction is the point at which evolution takes place. It is implemented by two genetic operations in this genetic algorithm: *crossover* and *mutation*. Crossover is a recombination operator which mixes the genes from two parents to reproduce two offsprings. Mutation is another recombination operator to randomly change an allel of an individual for keeping diversity in a population. In order to reproduce better offsprings, roulette selection strategy [50] is adopted for selecting parents for reproduction. This selection strategy makes sure that the fitter individuals have more chances to be selected for reproducing fitter offspring in the next generation.

In each generation, this genetic algorithm calculates fitness for all individuals and retains the fittest one. The fittest one then is further evolved by using a hill-climbing operator. The evolved fittest individual is compared with the evolved fittest one produced in the previous generation, and the fitter one is kept as the fittest individual in history. Hence, when the genetic algorithm terminates, the fittest individual in history is considered as the optimal switching graph. The genetic algorithm is described in Algorithm 5.2.

In Algorithm 5.2, P_{old} and P_{new} are old generation and new generation respectively, i_{best} retains the fittest individual in the history, $Cal\_Fitness(i)$ is a procedure of calculating the fitness of an individual i, $Best\_Individual(P)$ finds the fittest individual in the population P, $Select(P)$ is a genetic operator which is used for

selecting an individual among the population P in the roulette selection strategy, $Crossover(p_\alpha, p_\beta, o_\alpha, o_\beta, p_{crossover})$ is a knowledge-based recombination operator to reproduce two offsprings o_α and o_β from their parents p_α and p_β with the probability $p_{crossover}$, $Mutate(o_\beta, p_{mutation})$ is another recombination operator used for mutation with the probability $P_{mutatation}$, $Hill\_Climbing(i)$ is a knowledge-based operator which produces the local optimum from the individual i . $MaxGen$ and $PopSz$ represent the maximal number of generations and population size, respectively.

In the following sections, we will discuss in detail the issues involved in the genetic algorithm, which include coding (encoding and decoding), fitness function, genetic operators, hill-climbing operator, generation of initial population, and parameter setting.

Algorithm 5.2 Genetic Algorithm for Switching Graph Problem

create initial population P_{old};
for $\forall i \in P_{old}$, $Cal\_Fitness(i)$;
$i_{best} = Best\_Individual(P_{old})$;
$i_{best} = Hill\_Climbing(i_{best})$;
for generation $= 1$ *to* $MaxGen$ *do*
begin
  for $i = 1$ *to* $\lfloor PopSz/2 \rfloor$ *do*
  begin
    $P_{new} = \phi$;
    $p_\alpha = Select(P_{old})$;
    $p_\beta = Select(P_{old})$;
    $Crossover(p_\beta, o_\alpha, o_\beta, p_{crossover})$;
    $Mutate(o_\alpha, p_{mutation})$;
    $Mutate(o_\beta, p_{mutation})$;
  end
  $P_{old} = P_{new}$;
  for $\forall i \in P_{new}$, $Cal\_Fitness(i)$;
  $t_{best} = Best\_Individual(P_{new})$;
  $t_{best} = Hill\_Climbing(t_{best})$;
  if $Cal\_Fitness(i_{best}) < Cal\_Fitness(t_{best})$ *then*
    $i_{best} = t_{best}$;
end
output i_{best}.

5.5 Coding

Coding is a mechanism of building a link between the solutions to a problem and the chromosomes of a genetic algorithm. It links a genetic algorithm to the problem to be solved. Although the technique for coding may vary from a problem to another, there are two basic principles for choosing a coding: *the principle of meaningful*

building blocks and *the principle of minimal alphabets* [36].

The principle of meaningful building blocks is simply stated as: **the user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.**

The principle of minimal alphabets is simply stated as: **the user should select the smallest alphabet that permits a natural expression of the problem.**

We choose our coding for this genetic algorithm based on the two basic principles. There are two issues involved in the coding: *encoding* and *decoding*. The encoding transforms the solutions to the switching graph problem into the chromosomes of this genetic algorithm, while the decoding, in contrast, transforms the chromosomes of this genetic algorithm back to the solutions to the switching graph problem. The following section presents the encoding and decoding schemes used in this genetic algorithm.

5.5.1 Encoding scheme

According to Theorem 5.1, a switching graph represents a feasible layer assignment. Thus, a chromosome in this genetic algorithm corresponds to a switching graph. Suppose that $G = (V, E)$ is a LAP graph and $S_G(C)$ is a switching graph of G, where $V = V_{cluster} \cup V_{via}$, and $C \subseteq V_{cluster}$. Let $n = |V_{cluster}|$ and $p = |V_{via}|$. Then, $S_G(C)$ is represented as a binary string of n bits,

$$b_1 b_2 \cdots b_n,$$

where,

$$b_i = \begin{cases} 1 & \text{if } c_i \in C \\ 0 & \text{if } c_i \notin C \end{cases}$$

Under this encoding scheme, the LAP graph G (a special switching graph with

$C = \phi$) is always encoded as $00\cdots0$. For examples, the LAP graph shown in Figure 5.5 and the switching graph $S_G(\{c_3, c_6\})$ of the LAP graph shown in Figure 5.6 are encoded as 0000000 and 0010010, respectively.

However, this encoding scheme needs to be improved before being put into use because the defining length of some low-order potential schemata could be very long, and therefore those potential schemata have little chance to survive. This can be better understood by investigating a schema of the switching graph problem shown in Figure 5.5. Let's look at the schema $1*****1$. It is not difficult to see that this schema represents those switching graphs whose the direction of the arcs incident to clusters c_1 and c_7 has been reversed in relation to the LAP graph G. If the direction of the arcs is reversed, then the in-degree of the vertex v_1 becomes zero. Thus, the schema stands for the switching graphs in which the in-degree of the vertex v_1 is zero. In other words, the schema represents the switching graphs in which the vertex v_1 is a non-via vertex. Figure 5.9 illustrates the schema. In this figure, the dotted lines stand for arcs whose direction we do not care. Thus, this low order schema $(o(1*****1) = 2)$ is expected to pass on from a generation to next because it will contribute to finding a switching graph with minimum number of via vertices. However, the schema will be broken during crossover no matter where the crossover point is because of its long defining length, i.e. $\delta(1*****1) = 6$.

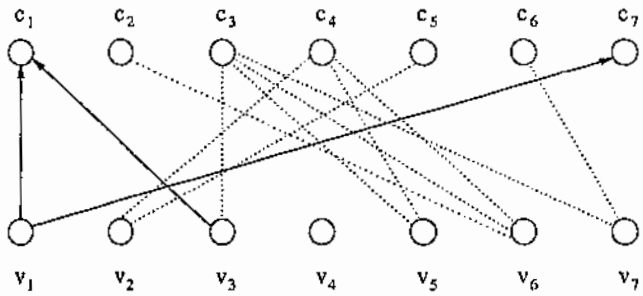


Figure 5.9: A schema

A good encoding scheme should be able to identify those low-order and high-performance schemata and manage to arrange them properly such that the average defining length is as short as possible. Based on this consideration, we develop a heuristic procedure to identify low-order and high-performance schemata and adjust the permutation for the genes such that the average defining length of the schemata is as short as possible.

It is not difficult to understand that a schema which corresponds to the situation that there is no via introduced at a via candidate vertex is low-order and high performance. For each via candidate vertex whose degree is not zero v_j, there are two such schemata. Algorithm 5.3 finds the two schemata S and S' that are associated with v_j.

Algorithm 5.3 Determining Schemata

determine the cluster vertices which are associated with v_j, *and put them into* C;
for $i = 1$ *to* n *do*
begin
  if $c_i \in C$ *and* $< c_i, v_j > \in E$ *then*
    $S[i] = '1'$;
    $S'[i] = '0'$;
  else
    if $c_i \in C$ *and* $< v_j, c_i > \in E$ *then*
      $S[i] = '0'$;
      $S'[i] = '1'$;
    else
      $S[i] = '*'$;
      $S'[i] = '*'$;
end
output S *and* S'.

In Algorithm 5.3, the parameters n and p are the numbers of the cluster vertices and

via candidate vertices respectively. Given a switching graph problem (a LAP graph) we apply Algorithm 5.3 on the via candidate vertices one by one and therefore can find $2 * p$ schemata.

Below are those schemata for the switching graph shown in 5.5 under the encoding scheme. It can be calculated that under the encoding scheme the average defining length is 2.5.

$s_1 = 1 * * * * * 1;$
$s_2 = 0 * * * * * *0;$
$s_3 = * * *10 * *;$
$s_4 = * * *01 * *;$
$s_5 = 1 * 0 * * * *;$
$s_6 = 0 * 1 * * * *;$
$s_7 = * * 10 * **;$
$s_8 = * * 01 * **;$
$s_9 = *100 * **;$
$s_{10} = *011 * **;$
$s_{11} = * * 0 * *0*;$
$s_{12} = * * 1 * *1*.$

Having found the low-order and high performance schemata, we use some heuristic rules to adjust the position of the genes which are associated with those long defining length schemata, such that the average defining length of the schemata is as short as possible. For instance, $b_7 b_1 b_6 b_3 b_4 b_2 b_5$ is an improved encoding. Under this encoding schema, the schemata become:

$s_1 = 11 * * * **;$
$s_2 = 00 * * * **;$
$s_3 = * * * * 1 * 0;$
$s_4 = * * * * 0 * 1;$
$s_5 = *1 * 0 * **;$
$s_6 = *0 * 1 * **;$
$s_7 = * * *01 * *;$
$s_8 = * * *10 * *;$
$s_9 = * * *001*;$
$s_{10} = * * *110*;$
$s_{11} = * * 00 * **;$
$s_{12} = * * 11 * **.$

Hence the average defining length is reduced to 1.5.

5.5.2 Decoding scheme

In contrast to the encoding scheme, the decoding scheme transforms a chromosome back to a switching graph. The decoding scheme is used in calculating the fitness of a chromosome and knowledge-based crossover. Two data structures are used in the decoding scheme. The first data structure is a hash table which is created during the encoding and used for mapping genes of a chromosome to clusters. The hash table for the encoding scheme $b_7b_1b_6b_3b_4b_2b_5$ is shown in Figure 5.10. It means that the first gene of a chromosome corresponds to the seventh cluster, and the second gene stands for the first cluster, and so on.

7	1	6		4	2	5

Figure 5.10: Hash table

Another data structure is a matrix *Template* which is used for storing a LAP graph G. Suppose that G has n cluster vertices and p via candidate vertices. Then G is represented in a $n \times p$ matrix *Template* which is defined as below:

$$Template[i][j] = \begin{cases} 1 & \text{if} < c_i, v_j > \in E; \\ -1 & \text{if} < v_j, c_i > \in E; \\ 0 & \text{otherwise.} \end{cases}$$

For example, the LAP graph shown in Figure 5.5 is represented as the matrix shown in Figure 5.11.

$$Template = \begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Figure 5.11: Template

Given a chromosome $s = a_1 a_2 \cdots a_n$, its corresponding switching graph M can be obtained by Algorithm 5.4. For example, the switching graph of the chromosome $s = 0011000$ is decoded as the above matrix shown in Figure 5.12.

Algorithm 5.4 Decoding

for $i = 1$ *to* n *do*

  for $j = 1$ *to* p *do*

    $M[i][j] = Template[i][j];$

for $i = 1$ *to* n *do*

  if $Hash(b_i) = 1$ *then*

    for $j = 1$ *to* p *do*

      $M[Hash[i]][j] = -M[Hash[i]][j];$

$$\begin{bmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Figure 5.12: The switching graph of s

5.6 Fitness Function

The objective of this genetic algorithm is to find a switching graph of a given LAP with minimum number of via candidate vertices. The fewer via candidate vertices a switching graph has, the fitter the corresponding chromosome is. Thus, the fitness of a chromosome i is defined in Equation 5.1:

$$f(i) = p - via(i) \tag{5.1}$$

where p is the number of via candidate vertices and $via(i)$ is the number of via vertices in the corresponding switching graph of i. The fitness of a chromosome i is calculated in three steps:

1. Decode the chromosome i to obtain its corresponding switching graph M;
2. Calculate $via(i)$ from M;
3. Calculate $f(i)$ by Equation 5.1.

5.7 Genetic operators

As in other conventional genetic algorithms, the developed algorithm uses three genetic operators: *selection*, *crossover*, and *mutation*.

5.7.1 Selection

The selection strategy is responsible for choosing the mates among the individuals in the population P_{old}. The selection approach used in this genetic algorithm is *roulette selection* [50]. The roulette selection assigns a probability to each individual i, computed as the proportion

$$F(i) = \frac{f(i)}{\sum_{k=1}^{MaxSz} f(k)}.$$

As a result, this genetic algorithm gives more reproductive chances, on the whole, to those individuals that are the fittests. The effect of the roulette wheel parent is to return a randomly selected parent. Although this selection procedure is random, each parent's chance of being selected is directly proportional to its fitness. On balance, over a number of generations this genetic algorithm will drive out the least fit individuals from the population.

5.7.2 Crossover

Crossover is a recombination operator which mixes the genes from two parents to produce two offspring. A common crossover is one-point conventional crossover. One-point crossover swaps two parent chromosomes at a randomly selected point, creating two offsprings. The one-point crossover is described as below:

Suppose that $p_1 = x_1x_2\cdots x_ix_{i+1}x_{i+2}\cdots x_n$ and $p_2 = y_1y_2\cdots y_iy_{i+1}y_{i+2}\cdots y_n$ are two selected parents, and j is a randomly selected point. Then, the following two offspring are generated as a result of one point crossover and replace their parents in the next generation: $o_1 = x_1x_2\cdots x_jy_{j+1}y_{j+2}\cdots y_n$ and $o_2 = y_1y_2\cdots y_jx_{j+1}x_{j+2}\cdots x_n$.

5.7.3 Mutation

Mutation is to randomly change an allele of an individual from one alphabet value to another in order to keep diversity in the generations. Since binary alphabet is used over the constrained via minimization problem, the original allele is exchanged for its complement.

The mutation operator offers the opportunity for new genetic material to be introduced into a population. From the theoretical perspective, it assures that given any population, the entire search space is reachable. The new genetic material does not originate from parents and is not introduced into their children by crossover.

5.8 Hill-climbing

The hill-climbing operator is used to improve the fitness of an individual. As its name implies, the operator is based on hill climbing technique [9].

Hill climbing is a search method for finding a maximum (or minimum) of an evaluation function. It considers the local neighborhood of a node, evaluating each neighbor node, and next examines those nodes with largest (or smallest) values.

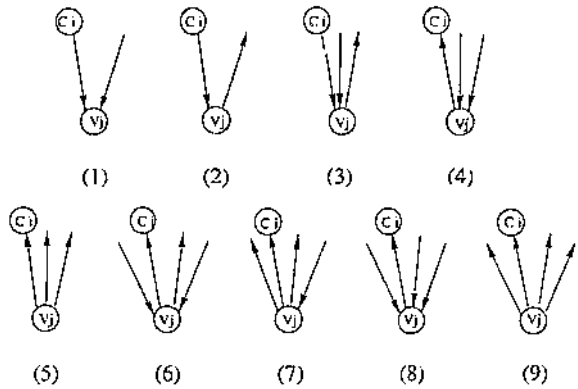


Figure 5.13: Types of the associating situations c_i and v_j

Unlike other search strategies that use evaluation functions (e.g., A^* algorithm) hill climbing is an irrevocable scheme: it does not permit us to shift attention back to previously suspended alternatives [9]. Compared with other search methods, the hill climbing method is simple in its computation and requires very little memory since alternatives do not need to be retained for future consideration.

In examining LAP graph we observed that the associating situations between a cluster vertex c_i and a via candidate vertex v_j can be categorized into the nine types shown in Figure 5.13, excluding isomorphic situations. By isomorphic situations we mean the patterns obtained by reversing the direction of all the arcs in the graph.

A weight $w(c_i, v_j)$ is assigned to each of the nine associating types. The weight is a measurement of the contribution to eliminating the via at the via candidate v_j if switching the cluster c_i. By switching a cluster we mean reversing the direction of the arcs incident to the cluster vertex. The weight $w(c_i, v_j)$ is defined in Equation 5.2.

$$w(c_i, v_j) = \begin{cases} -1 & \text{types (1), (5) and (9)} \\ 0 & \text{types (3), (6) and (7)} \\ 1 & \text{types (2), (4) and (8)} \end{cases} \tag{5.2}$$

where $< c_i, v_j > \in E$ or $< v_j, c_i > \in E$. The weight of a cluster vertex c_i, $W(c_i)$ is defined in Equation 5.3.

$$W(c_i) = \sum_{\forall < c_i, v_j > \in E} w(c_i, v_j) + \sum_{\forall < v_j, c_i > \in E} w(c_i, v_j) \tag{5.3}$$

Theorem 5.2. Assume that $G_1 = (V_{cluster} \bigcup V_{via}, E)$ is a LAP bigraph, $c \in V_{cluster}$ and $G_2 = S_{G_1}(\{c\})$. Let $Via(G_1)$ and $Via(G_2)$ be the numbers of via vertices in G_1 and G_2, respectively. Then, $Via(G_1) - Via(G_2) = W(c)$.

Proof. The via candidate vertices in the LAP graph G_1 can be partitioned into seven categories shown in Figure 5.14, if the isomorphic situations are excluded.

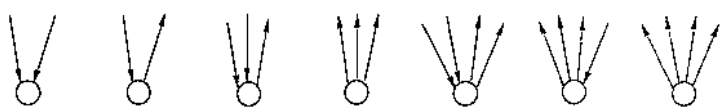


Figure 5.14: Categories of via candidate vertices

Assume that the numbers of the k^{th} category via candidate vertices on G_1 and G_2 are m_k and n_k respectively, and that r_q is the number of the q^{th} type via candidate vertices adjacent to c, where $1 \leq k \leq 7$, $1 \leq q \leq 9$. Then

$$\begin{aligned} Via(G_1) &= m_2 + m_3 + m_5 + m_6 \\ Via(G_2) &= n_2 + n_3 + n_5 + n_6 \\ W(c) &= -r_1 + r_2 + r_4 - r_5 + r_8 - r_9 \end{aligned}$$

Since

$$\begin{aligned} n_2 &= m_2 + (r_1 - r_2) \\ n_3 &= m_3 + (r_5 - r_4) \\ n_5 &= m_5 + (r_7 - r_6) \end{aligned}$$

Thus

$$\begin{aligned} Via(G_1) - Via(G_2) &= (m_2 + m_3 + m_5 + m_6) - (n_2 + n_3 + n_5 + n_6) \\ &= (m_2 + m_3 + m_5 + m_6) - \{[m_2 + (r_1 - r_2)] \\ &\quad + [m_3 + (r_5 - r_4)] + [m_5 + (r_7 - r_6)] \\ &\quad + [m_6 + (r_6 - r_7 - r_8 + r_9)]\} \\ &= -r_1 + r_2 + r_4 - r_5 + r_8 - r_9 \\ &= W(c). \quad \square \end{aligned}$$

Corollary 5.4. Assume that $G_1 = (V_{cluster} \cup V_{via}, E)$ is a LAP bigraph, $c \in V_{cluster}$ and $G_2 = S_{G_1}(\{c\})$, then

I. If $W(c) > 0$, then $Via(G_1) > Via(G_2)$;

II. If $W(c) = 0$, then $Via(G_1) = Via(G_2)$.

It can be seen from Corollary 5.4 that if we switch the cluster vertex c_i in a LAP graph (it is also a switching graph), we obtain a switching graph and the number of via vertices in the switching graph is reduced by $W(c_i)$, i.e. if $W(c_i)$ is positive then the number of via vertices in the switching graph will be decreased by $W(c_i)$; but if $W(c_i)$ equals to zero then the number of via vertices will not change. Therefore, a switching graph can be gradually improved by repeatedly selecting a cluster vertex c_i whose $W(c_i)$ is not negative, and then switching the cluster vertex. The procedure of 'selecting-and-switching' is repeated until no a such cluster vertex can be found. In this manner, a switching graph with fewer number of via vertices

can be obtained. This is the basic idea behind the hill-climbing operator. The hill-climbing operator is described in Algorithm 5.3.

Algorithm 5.5 Hill Climbing Operator

decode the chromosome i_{best} to obtain a switching graph G;
select a cluster vertex c_i in $V_{cluster}$ such that its $W(c_i)$ is maximum;
while $(W(c_i) \geq 0)$ do
begin
  reverse the direction of the arcs that are incident to c_i;
  select a cluster vertex c_i in $V_{cluster}$ such that its $W(c_i)$ is maximum;
end
encode G to create a chromosome i_{best} and output it.

5.9 Generation of initial population

In our preliminary research on this genetic algorithm, the initial population was created by randomly generating a collection of chromosomes. However, it was found that the genetic algorithm could not find an optimum for most cases. In fact, the solution found by the genetic algorithm was even worse than the solution found by the hill-climbing algorithm we developed based on the same model [106]. Further studies revealed that the cause for the problem was that the initial population did not cover enough representatives of hyperplanes. This led to a local optimum.

As the first trial to resolve the problem, the size of population was increased 10 times. However, the experimental results showed that the quality of the solutions were not significantly improved. In fact it made no difference for the large size problems because the 10 times increased population still made up a very small portion of the total number of the switching graphs. Next, we tried to increase the possibility of mutation ($P_{mutation}$) in order to introduce more diverse chromosomes

into the population. However, the experimental results revealed that the genetic algorithm performed a random search.

The effective method to overcome the problem is to utilize the domain knowledge to create the initial population such that good genes are contained in the initial population. Thus, the first and most important issue is to determine good schemata. Then, we use those schemata as templates to generate chromosomes. In order to better understand the process of the generation of the initial population we use an example to demonstrate how to create chromosomes from a schema.

Suppose $s = *1*0***$ is a schema. In our genetic algorithm, we generate 4 chromosomes based on each schema. What is actually done is to fix the alleles in defined positions and randomly generate the alleles for the undefined. For example, the chromosomes 0100111, 1100010, 0110000 and 0100101 may be the chromosomes generated in the initial population. It can be seen that in these chromosomes the second and forth alleles are 1 and 0, respectively.

Using the method stated in the coding we can build $2 * p$ schemata, where p is the number of the via candidate vertices in the switching graph. For each schema we create 4 chromosomes. Therefore, we create $8 * p$ chromosomes in the initial population.

5.10 Investigation of Parameters

5.10.1 Population size

The population size *PopSz* affects both the ultimate performance and efficiency of a genetic algorithm. Genetic algorithms generally do poorly with very small populations [37], because the population provides an insufficient sample size for most hyperplanes. A large population is more likely to contain representatives from a large number of hyperplanes. Hence the genetic algorithm can perform a

more informed search. As a result, a large population discourages premature convergence to suboptimal solutions. On the other hand, a large population requires more evaluations per generation, possibly resulting in an unacceptably slow rate of convergence.

It is observed that for this switching graph problem the number of representatives is closely related to the number of via candidate vertices p, which may be as small as 0, and as large as hundreds. Thus, the number of hyperplanes in different problems varies dramatically from one problem to another. Considering this factor, we adopt a variable population size rather than a constant population size in this genetic algorithm. The population size $PopSz$ is defined as a function of the variable p shown in Equation 5.4.

$$PopSz = 8 \times p \tag{5.4}$$

5.10.2 Maximal generations

As a result of performing a large number of experiments, we observed that most of the best solutions came from the first thirty generations. Therefore, we set the maximal generations ($MaxGen$) to 30 in this genetic algorithm.

5.10.3 Probabilities

The crossover probability $p_{crossover}$ controls the frequency with which the crossover is applied. In each new population, $PopSz*p_{crossover}$ individuals undergo crossover. The higher the corssover probability is, the quicker new individuals are introduced into the population. If the corssover probability is too high, many high performance schemata are destroyed. If the crossover probability is too low, the search may stagnate due to the lower exploration rate.

Another probability is on mutation. After the selection, each allele of each individual in the new population undergoes a random change with a probability $P_{mutation}$. Hence, approximately $PopSz * p_{crossover} * p_{mutation}$ mutations occur per generation. A low level of mutation serves to prevent any given allele from remaining forever converged to a single value in the entire population. A high level of mutation yields an essentially random search.

After a large number of experiments, we found that the following settings for probabilities of crossover and mutation are suitable for the developed genetic algorithm: $P_{crossover} = 0.95$; $P_{mutation} = 0.05$.

5.11 Implementation and Experiments

A program called KBGA was developed using C programming language to implement the proposed genetic algorithm on a Pentium 200 personal computer. KBGA contains about 3000 lines source code.

In order to test the performance of the genetic algorithm, we applied KBGA on the switchbox routing solutions obtained by our switchbox router HSR (see Figure 4.16 to Figure 4.22). For all tested problems, KBGA found the optimum solutions. Figure 5.15 to Figure 5.21 show those optimized results obtained by KBGA. The statistical results in term of the number of vias introduced before and after the constrained via minimization are shown in Table 5.1.

In order to further test the genetic algorithm, we developed a program to randomly generate switching graph problems. For each problem we used three different approaches: optimal algorithm, hill-climbing algorithm [106] as well as this genetic algorithm. The results are shown in Table 5.2. From Corollary 5.2 we know that the computational complexity of the optimal algorithm is exponential. It could take several months or so for the optimal algorithm to find the optimal solution for a large-size problem. As a result, we could not figure out the optimal solutions for those larger-size problems. For those problems we put '-' in the columns vias

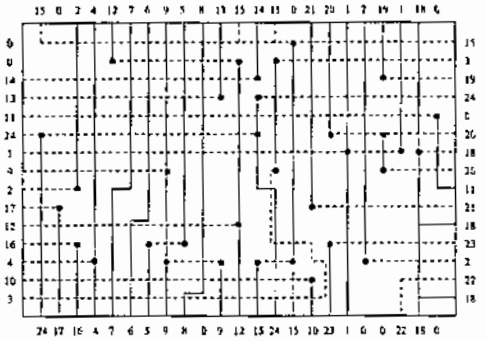


Figure 5.15: The solution to 'difficult' switchbox routing problem

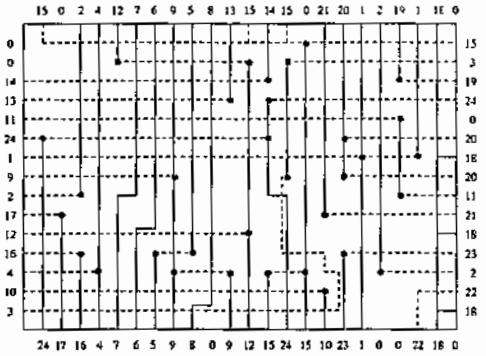


Figure 5.16: The solution to 'more difficult' switchbox routing problem

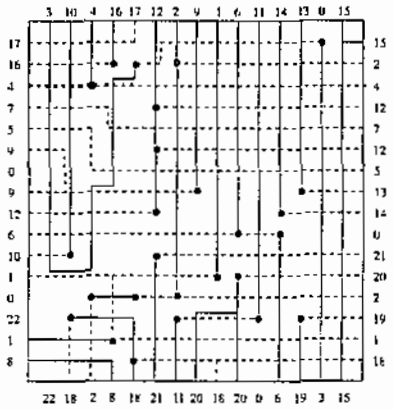


Figure 5.17: The solution to 'pedagogical' switchbox routing problem

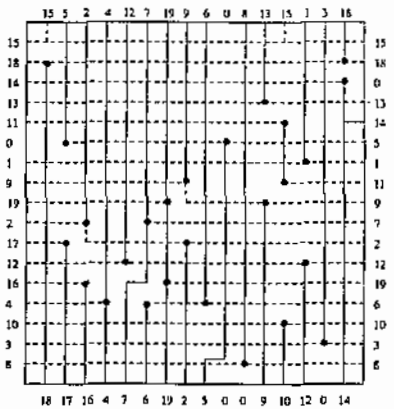


Figure 5.18: The solution to 'modified dense' switchbox routing problem

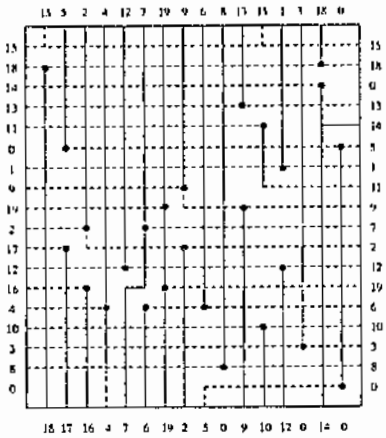


Figure 5.19: The solution to 'augmented dense' switchbox routing problem

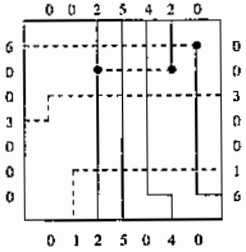


Figure 5.20: The solution to 'sample' switchbox routing problem

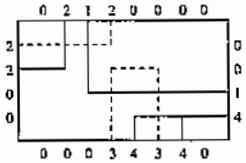


Figure 5.21: The solution to 'rectangle' switchbox routing problem

Problems	*No. of vias before CVM*	*No. of vias after CVM*
difficult	41	34
more difficult	40	33
pedagogical	34	26
modified dense	29	26
augmented dense	31	26
sample	7	3
rectangle	7	0

Table 5.1: Statistics on the numbers of vias introduced before and after constrained via minimization for the switchbox routing solutions

and time.

It can be seen from Table 5.2 that KBGA successfully found the optimum solutions for most of the randomly generated problems, and that the computational time ranged from 1 to 9 seconds. Although it cannot be guaranteed that KBGA can find the optimum solutions for all switching graph problems, KBGA can find the optimum solutions in most cases and the computational time is in the order of only seconds.

Problem	*Characteristics*			*Optimal*		*Hill-Climbing*		*Genetic*	
Index	n	p	*vias*	*vias*	*time(sec)*	*vias*	*time(sec)*	*vias*	*time(sec)*
1	12	9	7	2	9	2	0.0002	2	1
2	12	16	13	7	9	8	0.0004	7	1
3	14	21	13	7	34	7	0.0005	7	2
4	58	10	9	-	-	3	0.0012	3	1
5	34	14	12	-	-	2	0.0013	2	5
6	28	30	17	10	150842	10	0.0019	10	5
7	13	49	37	24	17	25	0.0011	24	8
8	11	28	23	15	4	17	0.0017	15	4
9	16	22	15	7	136	8	0.0008	7	2
10	12	11	7	1	9	3	0.0002	1	1
11	18	21	13	9	493	9	0.0006	9	2
12	26	25	18	8	11098	11	0.0014	8	4
13	48	19	16	-	-	6	0.0024	4	3
14	30	12	14	3	244574	3	0.0010	3	2
15	38	16	9	-	-	3	0.0012	1	3
16	38	20	10	-	-	6	0.0013	5	3
17	43	20	13	-	-	5	0.0022	4	4
18	16	42	29	20	140	22	0.0011	20	7
19	20	42	31	18	1505	22	0.0022	18	8
20	17	27	17	9	274	9	0.0009	9	4
21	16	9	5	1	133	3	0.0030	1	1
22	13	19	26	17	18	17	0.0010	17	6
23	25	25	20	8	6081	10	0.0017	8	4
24	22	37	24	16	5144	16	0.0019	16	7
25	10	19	9	7	2	9	0.0002	7	2
26	19	23	14	7	1096	10	0.0008	7	3
27	19	28	18	11	767	12	0.0009	11	4
28	23	28	19	11	8013	12	0.0014	11	5
29	28	28	19	8	92699	8	0.0019	8	5
30	8	12	9	4	1	5	0.0002	4	3
31	42	16	13	-	-	5	0.0015	2	2
32	30	13	5	2	285311	2	0.0006	2	1
33	51	15	9	-	-	2	0.0018	1	2
34	17	10	10	2	48	2	0.0004	2	2
35	55	18	16	-	-	3	0.0037	3	4
36	14	58	34	28	6	28	0.0013	28	9

Table 5.2: Experimental results for the randomly generated switching graph problems

5.12 Summary

In this chapter, we proposed a new graph-theoretic model, namely switching graph model, for the constrained via minimization problem. Then on the basis of the model, we presented the first genetic algorithm for the constrained via minimization.

The switching graph model can handle both grid-based and gridless routing problems, and allows arbitrary wire segments split at a via candidate, hence it is practical. In addition, the model can cope with the irrational situations that might be raised by designers' interference, thus it is robust. Moreover, the model can be configured to meet some practical requirements, therefore it is flexible.

The developed genetic algorithm based on the switching graph model is effective since it can find the optimum solution in most cases, and it is efficient because it produces the optimum solution in a small number of generations. These merits of the genetic algorithm benefited from:

- The good encoding scheme. The encoding scheme used in the genetic algorithm is a natural expression of the problem and used the smallest alphabet, i.e. $\{0, 1\}$. Besides, under the encoding scheme any randomly generated or reproduced chromosomes are always legible ones and thus the genetic algorithm does not need to check whether a newly-generated chromosome is a valid or not, which takes a significant amount of processing time. The last but the most important is that under the encoding scheme the defining length of the potential low-order and high-performance scheme is short. As a result, good schemata can survive during the process of evolution and therefore contribute to the production of the optimum solution.

- The knowledge-based method to generating initial population. Chromosomes created in the knowledge-based method contain many good genes and cover much more representatives of hyperplanes in comparison to those created in conventional methods.

- The hill-climbing operator. The hill-climbing operator can further improve the fitness of a chromosome.

Chapter 6

Conclusions

In this thesis we have investigated intelligent approaches to VLSI routing problems, with the aim of developing more effective and efficient VLSI routing methods and techniques. We divided the problems challenging today's VLSI routing tools into three categories: the problems whose size is massive, the problems which cannot be modeled, and the problems whose computational complexity is NP-complete or NP-hard. We studied intelligent approaches to those three category problems by investigating three typical problems: the 3-D shortest path connection problem, the switchbox routing problem and the constrained via minimization, each of which belongs to one of the three categories.

This chapter summaries the contributions of this thesis, emphasizes the significance of the intelligent approaches to VLSI routing, and outlines the future research work.

6.1 Contributions of This Thesis

This thesis has presented three new intelligent approaches to three typical problems in VLSI routing. In Chapter 3 we proposed a new A^* algorithm approach to the 3-D shortest path connection problem. In Chapter 4 we presented a new hybrid approach to the switchbox routing problem. In Chapter 5 we presented the first

genetic algorithm approach to the constrained via minimization problem. These are the major contributions of this thesis.

For the 3-D shortest path connection problem, we proposed a new A^* algorithm approach which uses an economical representation and a novel backtrace and clearance technique. The A^* algorithm approach transforms the shortest path connection problem into a state space problem of AI, and then uses a modified A^* algorithm search technique to solve the state space problem. The A^* algorithm contains two consequent procedures: exploration, and backtrace & clearance. Starting from the initial state s, which corresponds to one net terminal, the exploration procedure uses a best-search technique to search goal state t, which corresponds to another net terminal. Once t is reached, the novel backtrace & clearance procedure traces back the shortest path among the states that have been traversed and clears the marks made during the exploration. Furthermore, the thesis presented three variations of the A^* algorithm. The first variation is used for finding the shortest path connection between two sub-wires. The second variation is used for finding the shortest path connection with minimization of layer changes. The third variation is used for finding the shortest path connection with minimization of crosstalk which is used in performance-driven VLSI routing. Compared with the other shortest path connection algorithms, this A^* algorithm approach has the following advantages :

- It can guarantee to find a path between a pair of terminals of a net if one exists, and the path found is the shortest. The guarantee of finding an optimal solution is one of the major concerns in VLSI routing.
- It can achieve high computational efficiency. This A^* algorithm method uses heuristic information to guide the search in the exploration phase and therefore reduces the search space. Since the search space is proportional to the computational time, this A^* algorithm method is computationally fast, especially when it is used in a routing environment where the routed nets are spare. It has been proven in Chapter 3 that the computational complexity is $O(n)$ at best.

- It has an economical representation. The representation used in this algorithm uses 1 bit to represent a grid point, while other path connection methods uses at least 2 bits for each grid point.

- It adopts a novel backtrace technique, which could be applied in the other path connection algorithms.

- It is flexible. This A^* algorithm method can be easily modified to meet different practical demands.

For the switchbox routing problem, we presented a new hybrid approach, combining knowledge-based and algorithmic routing techniques. One of our major contributions to the switchbox routing problem is the synchronized routing technique, a new knowledge-based routing technique. The other contributions we have made to the switchbox routing problem include the modified rip-up routing technique, the pattern routing techniques, as well as the integration of knowledge-based techniques and algorithmic routing techniques. This hybrid approach has the following features:

- It can benefit both the high routing completion rate of knowledge-based techniques and the high computation speed of algorithmic routing techniques.

- It can discover and make use of mutual constraints among the nets in a switchbox in the context, which benefits achieving high routing completion rate. In fact, it successfully found solutions to all the renowned benchmark problems.

- Its computational efficiency is remarkably high. It takes few seconds to find a solution to those tested benchmarks.

- It has a very high degree of parallelism and thus can be developed into a parallel switchbox to further improve the efficiency of switchbox routing.

For the constrained via minimization problem, we proposed a new graph-theoretic model, namely switching graph model. Then on the basis of the model, we presented the first genetic algorithm for the constrained via minimization.

The merits of the new graph-theoretic model are:

- It can handle both grid-based and gridless routing problems, and allows arbitrary wire segments split at a via candidate, hence it is practical.
- It can cope with some irrational situations that might be raised by designers' interference, thus it is robust.
- It can be configured to meet practical requirements, therefore it is flexible.

The developed genetic algorithm based on the switching graph model is effective since it can find the optimum solution in most cases, and it is efficient because it produces the optimum solution in a small number of generations. The merits of the genetic algorithm are:

- It has a good encoding scheme. The encoding scheme used in the genetic algorithm is a natural expression of the problem and uses the smallest alphabet, i.e. $\{0, 1\}$. Besides, under this encoding scheme any randomly generated or reproduced chromosomes are always legible ones and thus the genetic algorithm does not need to check whether a newly-generated chromosome is valid or not, which takes a significant amount of processing time. The last but the most important point is that under this encoding scheme the defining length of the potential low-order and high-performance scheme is short. As a result, good schemata can survive during the process of evolution and therefore contribute to the production of the optimum solution.
- It makes use of domain-specific knowledge to enhance its performance. The first place where domain-specific knowledge is used is the knowledge-based method for generating the initial population. Chromosomes created in the

knowledge-based method contain many good genes and cover much more representatives of hyperplanes in comparison to those created in conventional methods. The second place where domain-specific knowledge is applied is the hill-climbing operator. The hill-climbing operator can improve the fitness of a chromosome by producing a local optimum from a given chromosome, and most importantly, it can bring some good genes into a population.

All approaches proposed in this thesis have been implemented and tested. The theoretical analysis and experimental results have shown that the proposed approaches are not only computationally fast but also perform close to the optimal levels required.

6.2 Intelligent Approaches to VLSI Routing

In studying VLSI routing problems we found that many aspects of VLSI routing lend themselves to intelligent approaches. For example:

- Many problems in VLSI routing can be thought of as a search over a huge solution space. Any blind search of this space is impractical because of the time required. The search process must therefore be guided by domain-specific information. Intelligent search is a good way to solve this category of problems.

- Although some problems in VLSI routing can be solved by using exact algorithms, there are a significant number of problems in VLSI routing which do not yield an algorithmic solution because they cannot be well represented in a mathematical model. Solving these problems is often based on routing knowledge and expertise. Thus, making use of domain-specific knowledge and expertise is extremely important in solving those problems which do not yield an algorithmic solution.

- Many problems in VLSI are complex combinatorial optimization problems. Most often, the goal of these optimization problems is finding an optimal or near-optimal solution in a reasonable time. Therefore, robust and efficient optimization approaches such as knowledge-based genetic algorithms play an important role in solving those VLSI routing optimization problems.

- Much of the VLSI routing is exploratory or evolutionary in style. Initially, the routing plan for a routing task is vague. As routing proceeds, connections are frequently modified. The design plan is not known until the routing is finished and it is not usually applicable in detail to the next routing problem. This is in contrast to conventional approaches, which are usually built around a sequence of operations that are only weakly influenced by the routing data. Hence, discovering and making good use of the constraints and interactions among the nets in the context are extremely important.

In conclusion, intelligent approaches are effective and efficient in solving VLSI routing problems.

6.3 Future Work

In this section we propose some research extensions which can follow from the findings of this thesis.

6.3.1 Future work on the A^* algorithm approach to 3-D shortest path connection problem

Recent research on path connection algorithms has shown that the solutions of the traditional maze routing algorithm can violate via-rules in a multi-layer routing environment, and that a straightforward extension to the maze algorithm that disallows via-rule incorrect routes may either cause a suboptimal route to be found, or more seriously, cause the failure to find any route even if one exists [25]. Thus

the following questions arise: is it possible to extend the A^* algorithm approach to a routing environment where more practical via-rules are applied? If so, then how should we represent a 3-D routing environment in which some practical via-rules are taken into account? How should we transform the path connection problem under the 3-D routing representation into a state space problem? How should we construct an additive heuristic function which meets the property of A^* algorithm? How should we modify the backtrace technique introduced in this thesis such that it can be applied in the new A^* algorithm? These issues are open for future research.

6.3.2 Future work on the hybrid approach to switchbox routing problem

Computation speed and routing completion rate are two major concerns in switchbox routing. With regard to the computation speed there is still room for improvement. A possible way to further improve the computation speed of the hybrid approach would be to develop the parallelism of the routing techniques used in the hybrid approach and implement the hybrid approach in a parallel processing environment. In fact, the routing techniques used in the hybrid approach have great potential to be extended to parallel algorithms. For example, in synchronized routing, which takes most time of a switchbox routing process, the sub-wires whose routing areas do not intersect each other can be extended in parallel.

Another possible way to improve the computation speed of the hybrid approach, and probably the routing completion rate, is to improve the order of extending unconnected sub-wires in the synchronized routing. The ordering strategy used in the hybrid approach is to choose an unconnected sub-wire by scanning the switchbox from left to right, and from top to bottom. A better ordering strategy might be developed which results in less rip-up and reroute. This would contribute to both high computation speed and high routing completion rate.

6.3.3 Future work on the genetic algorithm approach to constrained via minimization problem

The genetic algorithm approach to constrained via minimization problem is based on the switching graph model which transforms a constrained via minimization problem into a switching graph problem. A key step in transforming a constrained via minimization problem into a switching graph problem is the construction of a LAP graph for a given routing. In this thesis we have given a detailed discussion on how to construct a LAP graph for a grid-based routing, but we did not elaborate on how to construct a LAP graph for a gridless routing. Although the basic ideas behind building a LAP graph for a grid-based routing and a gridless routing are the same, building a LAP graph for a gridless routing is much more complicated than building a LAP graph for a grid-based routing because in a gridless routing the widths of wires might be different and the sizes of vias are not the same. Therefore, if we directly apply the method which we used in constructing a LAP graph for a grid-based routing on a gridless routing, it would lead to the violation of some design rules. Thus, there is a need to investigate how to construct a LAP graph for a gridless routing.

In addition, the unique power of genetic algorithms shows up with parallel computing. Parallel genetic algorithms with information exchange between searches are often more efficient than independent searches. Distributed genetic algorithms combine the speed of parallel computing and the advantage of inherent parallelism available in genetic algorithms. Thus, another interesting research is to extend the genetic algorithm to a distributed genetic algorithm to improve both the computation speed and the quality of solutions.

Bibliography

[1] K. Ahn and S. Sahni, Constrained via minimization, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 2, 1993, 273-282.

[2] S.B. Akers, A modification of Lee's path connection algorithm, *IEEE Trans. on Electronic Computers*, Vol. EC-16, 1967, 97-98.

[3] T. Back, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York, 1996.

[4] A. Barr and E.A. Feigenbaum, *The Handbook of Artificial Intelligence*, Vol. 1, Addison-Wesley, Reading, 1981.

[5] F. Barahona, On via minimization, *IEEE Trans. on Circuits and Systems*, Vol. CAS-37, No. 4, 1990, 527-530.

[6] H.B. Bakoglu, *Circuits, Interconnections, and Packaging for VLSI*, Addison-Wesley, Reading, 1990.

[7] H. Bollinger, A mature DA system for PC layout, *Proc. 1st International Printed Circuit Conference*, 1979, 85-99.

[8] B.P. Buckles and F. E. Petry, *Genetic Algorithms*, IEEE Computer Science Press, Los Alamitos, 1992.

[9] A. Bundy (ed), *Artificial Intelligence Techniques: A Comprehensive Catalogue*, Springer, Berlin, 1997.

[10] M. Burstein and R. Pelavin, Hierarchical wire routing, *IEEE Trans. on Computer-Aided Design*, Vol. CAD-2, No. 4, 1983, 223-234.

[11] R.W. Chan, Y. Kajitani, and S.P. Chan, A graph-theoretic via minimization algorithm for two-layer printed circuit boards, *IEEE Trans. on Circuits and Systems*, Vol. CAS-30, No. 5, 1983, 284-299.

[12] K.C. Chang and H.C. Du, A preprocessor for the via minimization problem, *Proc. 23rd ACM/IEEE Design Automation Conference*, 1986, 702-707.

[13] K.C. Chang and D.H. Du, Efficient algorithms for layer assignment problem, *IEEE Trans. on Computer-Aided Design*, Vol. CAD-6, No. 1, 1987, 67-78.

[14] K.C. Chang and H.C. Du, Layer assignment problem for three-layer routing, *IEEE Trans. on Computers*, Vol. C-37, No. 5, 1988, 625-632.

[15] L. Chambers (ed), *Practical Handbook of Genetic Algorithms*, CRC, Boca Raton, 1995.

[16] C.C. Chang and J. Cong, An efficient approach to multi-layer layer assignment with application to via minimization, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 5, 1999, 608-620.

[17] F.L. Chan, M.D. Spiller, and A.R. Newton, WELD-An environment for web-based electronic design, *Proc. 35th ACM/IEEE Design Automation Conference*, 1998, 146-151.

[18] R.W. Chen, Y. Kajitani, and S.P. Chan, A graph-theoretic via minimization algorithm for two-layer printed circuit boards, *IEEE Trans. on Circuits and Systems*, Vol. CAS-30, 1983, 284-299.

[19] J.S. Cherng, S.J. Chen, C.C. Tsai, and J.M. Ho, An efficient approach for via minimization in multi-Layer VLSI/PCB routing, *Proc. IEEE 1995 Custom Integrated Circuits Conference*, 1995, 473-476.

[20] T.W. Cho, S.S. Pyo, and J.R. Heath, PARALLEX: A parallel approach to switchbox routing, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 13, No. 6, 1994, 684-693.

[21] Y.-C. Chou and Y.-L. Lin, A graph-partitioning-based approach for multi-layer constrained via minimization, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1998, 426-429.

[22] M.J. Ciesielski and E. Kinnen, An optimal layer assignment for routing in IC's and PCB's, *Proc. 18th Design Automation Conference*, 1981, 733-737.

[23] G.W. Clow A global routing algorithm for general cells, *Proc. 21st ACM/IEEE Design Automation Conference*, 1984, 45-51.

[24] J.P. Cohoon and P.L. heck, BEAVER: A computational geometry based tool for switchbox routing, *IEEE Transaction on Computer-Aided Design*, Vol. CAD-7, 1983, 684-697.

[25] J. Cong, J. Fang, and K.-Y. Khoo, Via design rule consideration in multi-layer maze routing algorithms, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 2, 2000, 215-223.

[26] F. Curatelli, Switchbox routing with rerouting capabilities in VLSI design, *IEE Proceedings. G, Circuits, Devices, and Systems*, Vol. 137, No. 3, 1990, 210-.

[27] W.A. Dees and P. G. Karger, Automated rip-up and reroute techniques, *Proc. 19th Design Automation Conference*, 1982, 432-439.

[28] R. Diestel, *Graph Theory*, Springer, New York, 1997.

[29] R. Drechsler, *Evolutionary Algorithms for VLSI CAD*, Kluwer Academic Publishers, Boston, 1998.

[30] S.-C. Fang, K.-E. Feng, and W.-S. Chen, Constrained via minimization with practical considerations for multi-layer VLSI/PCB routing problems, *Proc. 28th ACM/IEEE Design Automation Conference*, 1991, 60-65.

[31] M. Feuer, VLSI design automation: An introduction, *Proc. of the IEEE*, Vol. 71, No. 1, 1983, 1-9.

[32] N. Funabiki and S. Nishikawa, A neural network model for multilayer topological via minimization in a switchbox, *IEEE Trans. on Computer-Aided Design of Integerated Circuits and Systems*, Vol. 15, No. 8, 1996, 1012-1020.

[33] T. Gao and C.L. Liu, Minimum crosstalk switchbox rotuing, Integration, the VLSI Journal, Vol. 19, No. 3, 1995, 161-180.

[34] W.Z. George, *Routing, Placement, and Partitioning*, Ablex Publishing Corporation, 1994.

[35] N. Gockel, R. Drechsler and B. Becker, A multi-layer detailed routing approach based on evolutionary algorithms, *Proc. 1997 IEEE International Conference on Evolutionary Computation*, 1997, 557-562.

[36] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, 1989.

[37] J.J. Gerfenstette, Optimization of control parameters for genetic algorithms, *IEEE Trans. on Systems, Man, and Cybernetics*, Vol. 16, No. 1, 1986, 122-128.

[38] Z. Hai and D.-F. Wong, Crosstalk-constrained maze routing based on Lagrangian relaxation, *Proc. International Conference on Computer Design VLSI in Computers and Processors*, 1997, 628-633.

[39] F.O. Hadlock, A shortest path algorithm for grid graphs, *Networks*, Vol. 7, 1977, 323-334.

[40] A. Hanafusa, Y. Yamashita, and M. Yasuda, Three-dimensioanl routing for multilayer ceramic printed circuit boards, *Proc. IEEE International Conference on Computer-Aided Design*, 1990, 386-389.

[41] P.E. Hart, N.J. Nilsson, and B. Rafael, A formal basis for heuristic determination of minimum cost paths, *IEEE Trans. on Sys. Sci. and Cyb.*, Vol. SSC-4, 1968, 100-107.

[42] S. Hartmann, M.W. Schiffer, and A.S. Schulz, Switchbox routing in VLSI design: Closing the complexity gap, *Theoretical Computer Science*, Vol. 203, No. 1, 1998, 31-49.

[43] A. Hashimoto and J. Stevens, Wire routing by optimizing channel assignment within large apparatus, *Proc. 8th Design Automation Workshop*, 1971, 155-163.

[44] S. Heiss, A path connection algorithm for multi-layer boards, *Proc. 5th Design Automation Workshop*, 1968, 6.1-6.14.

[45] W. Heyns, W. Sansen, and H. Beke, A line-expansion algorithm for the general routing problem with a guaranteed solution, *Proc. 17th Design Automation Conference*, 1980, 243-249.

[46] D.W. Hightower, A solution to line-routing problem on the continuous plane, *Proc. 6th Design Automation Workshop*, 1969, 1-24.

[47] D.W. Hightower, The interconnection problem - a tutorial, *Proc. 10th Design Automation Conference*, 1973, 1-21.

[48] J.H. Hoel, Some variations of Lee's algorithm, *IEEE Trans. on Computers*, Vol. C-25, 1976, 19-24.

[49] T.C. Hu and E.S. Kuh (ed), *VLSI Circuit Layout: Theory and Design*, IEEE Press, New York, 1985.

[50] J.H. Holand, *Adaptation in Natural and Artificial Systems*, Univ. of Michigan Press, 1975.

[51] R. Joobbani and D.P. Siewiorek, WEAVER: A knowledge-based routing expert, *IEEE Design & Test*, Vol. 3, No. 1, 1986, 12-23.

[52] P. Judea, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Canada, 1984.

[53] D.-C. Jun, S. Raje and M. Sarrafzadeh, Fast approximation algorithms on maxcut, k-coloring, and k-color ordering for VLSI applications, *IEEE Trans. on Computers*, Vol. 47, No. 11, 1998, 1253-66.

[54] Y. Kajitani, On via minimization of routing in a 2-layer board, *Proc. IEEE International Conference on Circuits and Computers*, 1980, 295-298.

[55] A. Kanasugi, T. Shimayama, N. Nakaya, and T. Iizuka, A genetic algorithm for switchbox routing problem, *Lecture Notes in Computer Science*, 1998, 247-254.

[56] S. Kirkpatrick, C.D. Gelatt and Jr., M.P. Vecchi, Optimization by simulated annealing, *Science*, Vol. 220, 1983, 671-680.

[57] R.E. Korf, Linear-space best-first search, *Artificial Intelligence*, Vol. 62, 1993, 41-78.

[58] Y.S. Kuo, T.C. Chern, and W.K. Shih, Fast algorithm for optimal layer assignment, *Proc. 25th ACM/IEEE Design Automation Conference*, New York, 1988, 554-559.

[59] C.Y. Lee, An algorithm for path connections and its application, *IRE Trans. on Electronic Computers*, Vol. EC-10, 1961, 346-365.

[60] C.H. Lee, Multilayer routing problem, *Progress in Computer-Aided VLSI Design, Vol. 2: Techniques*, edited by G.W. Zobrist, Ablex Publishing Corporation, USA, 1989, 93-121.

[61] J. Lienig and K. Thulasiraman, GASBOR: a genetic algorithm for solving the switchbox routing problem, *Journal of Circuits, Systems and Computers*, Vol. 6, No. 4, 1996, 359-373.

[62] J. Lienig, A parallel genetic algorithm for performance-driven VLSI routing, *IEEE Trans. on Evolutionary Computation*, Vol.1, No.1, 1997, 29-39.

[63] W.K. Luk, A greedy switchbox router, *Integration, The VLSI Journal*, Vol. 3, 1985, 129-149.

[64] N. Mani and B. Srinivason, Optimizing zig-zags in maze-routing algorithm, *Proc. 1998 International Conference on Systems, Man, and Cybernetics*, Vol. 4, 1998, 3949-3952.

[65] P. Mazumder and E.M. Rudnick, *Genetic Algorithms for VLSI Design, Layout & Test Automation*, Prentice-Hall, London, 1999.

[66] K. Mikami and K. Tabuchi, A computer program for optimal routing of printed circuit connectors, *Proc. of IFIP*, 1968, 1475-1478.

[67] P. Molitor, Constrained via minimization for systolic arrays, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 9, No. 5, 1990, 537-542.

[68] P. Molitor, A hierarchy preserving hierarchical bottom-up 2-layer wiring algorithm with respect to via minimization, *Integration, The VLSI Journal*, Vol. 15, No. 1, 1993, 73-96.

[69] E.F. Moore, The shortest path through a maze, *Annals of the Harvard Computation Laboratory*, Vol. 30, Pt. II, 1959, 185-292.

[70] G.E. Moore, Cramming more components onto integrated circuits, *Electronics Magazine*, Vol. 38, 1965, 114-117.

[71] N.J. Naclerio, S. Masuda, and K. Nakajima, Via minimization for gridless layouts, *Proc. Design Automation Conference*, 1987, 159-165.

[72] N.J. Naclerio, S. Masuda, and K. Nakajima, The via minimization is NP-complete, *IEEE Trans. on Computers*, Vol. C-38, No. 11, 1989, 1604-1608.

[73] A.S. Naveed, *Algorithms for VLSI physical design automation*, Kluwer Academic Publishers, Boston, 1993.

[74] J. Nievergelt and F. P. Preparata, Plane-sweep algorithms for intersecting geometric figures, *Commun. ACM*, Vol. 25, 1982, 739-747.

[75] N.J. Nilsson, *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

[76] T. Ohtsuki (Ed), *Layout Design and Verification*, Elsevier Science Publishers B.V., North-Holland, 1986.

[77] J.G. Ousterhout, R.M. Hamachi, W. Scott, and G. Taylor, Magic: A VLSI layout system, *Proc. 21st ACM/IEEE Design Automation Conference*, 1984, 152-159.

[78] M. Palczewski, Plane parallel A^* maze router and its application to FPGAs, *Proc. 29th ACM/IEEE Design Automation Conference*, 691-697.

[79] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, 1984.

[80] R.Y. Pinter, Optimal layer assignment for interconnect, *Proc. IEEE Circuits and Systems Conference*, 1982, 389-401.

[81] B.T Preas and M.J. Lorenzetti (ed), *Physical Design Automation of VLSI Systems*, The Benjamin/Cummings Publishing Company, 1988.

[82] D.A. Pucknell and K. Eshraghian, Basic VLSI Design, Prentice Hall, 1994.

[83] D. Richards, Complexity of single-layer routing, *IEEE Trans. on Computers*, Vol. C-33, No. 3, 1984, 286-288.

[84] R. River and C. Fiduccia, A greedy channel router, *Proc. 19th ACM/IEEE Design Automation Conference*, 1982, 418-424.

[85] F. Rubin, The Lee path connection algorithm, *IEEE Trans. on Computers*, Vol. C-23, No. 9, 1974, 907-914.

[86] F. Rubin, An iterative technique for printed wire routing, *Proc. 11th Design Automation Workshop*, New York, 1974, 308-313.

[87] S.M. Sait, *VLSI Physical Design Automation: Theory and Practice*, World Scientic, Singapore, 1999.

[88] M. Sarrafzadeh and D.T. Lee, *Algorithmic Aspects of VLSI Layout*, Word Scientific, Singapore, 1993.

[89] M. Sarrafzadeh, *An Introduction to VLSI Physical Design*, MaGraw Hill, New Yourk, 1996.

[90] W. Scott and J. Ousterhout, Plowing: Interactive stretching and compaction in Magic, *Proc. 21st ACM/IEEE Design Automation Conference*, 1984, 166-172.

[91] H. Shin, and A. Sangiovanni-Vincentelli, MIGHTY: A rip-up and reroute detailed 'Router', *Proc. IEEE International Conference on Computer-Aided Design*, 1986, 2-5.

[92] C.-J. Shi, Assigned hypergraph model of constrained via minimization, *Proc. 2nd Great Lakes Symposium on VLSI*, 1992, 159-166.

[93] C.J. Shi, A. Vanneli and J. Vlach, Performance-driven layer assignment for printed circuit boards and integrated circuits, *Proc. Ninth Annual IEEE International ASIC Conference and Exhibit*, 1996, 171-174.

[94] C.-J.R. Shi, Solving constrained via minimization by compact linear programming, *Proc. IEEE Asia and South Pacific Design Automation Conference*, 1997, 635-640.

[95] C.-J Shi, J.A. Brzozowski, A charaterization of signed hypergraphs and its applications to VLSI via minimization and logic synthesis, *Discrete Applied Mathematics*, Vol. 90, No. 1, 1999, 223-243.

[96] Semiconductor Industry Association, *National Technology Roadmap for Semiconductors*, 1997.

[97] J. Soukup, Fast maze algorithm, *Proc. 15th Design Automation Conference*, 1978, 100-102.

[98] J. Soukup, Circuit layout, *Proc. of the IEEE*, Vol. 69, No. 10, 1981, 1281-1304.

[99] J. Soukup, Maze router without a grid map, *IEEE/ACM International Conference Computer-Aided Design Gedist Dig. Tech. Papers*, 1992, 382-385.

[100] K.R. Stevens and W.M. VanCleemput, Global via elimination in generalized routing environment, *Proc. International Symposium on Circuits and Systems*, 1979, 689-692.

[101] W.H. Sung, A. Jagannathan and J. Lillis, Timing-driven maze routing, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 19, No. 2, 2000, 234-241.

[102] T.G. Szymanski, Dogleg channel routing is NP-Complete, *IEEE Trans. on Computer-Aided Design*, Vol. CAD-4, 1985, 31-41.

[103] K. Takahashi and T. Watanabe, A heuristic algorithm to solve constrained via minimization for three-layer routing problems, *Proc. 1998 IEEE International Symposium on Circuits and Systems*, 1998, 254-257.

[104] M. Tang, H.N. Cheung, and K. Eshraghian, A novel shortest path connection algorithm for multilayer VLSI routing, *Proc. 3rd Biennial Engineering Mathematics and Applications Conference*, Adelaide, 1998, 479-482.

[105] M. Tang, K. Eshraghian, and H.N. Cheung, A heuristic three-dimensional path connection algorithm for MCM routing, *Proc. 11th International Conference on Computers and Their Applications in Industry and Engineering*, Las Vegas, 1998, 26-29.

[106] M. Tang, K. Eshraghian, and H.N. Cheung, An efficient approach to constrained via minimization for two-layer VLSI routing, *Proc. IEEE Asia and South Pacific Design Automation Conference*, Hong Kong, 1999, 149-152.

[107] M. Tang, K. Eshraghian, and H.N. Cheung, A genetic algorithm for constrained via minimization, Proc. IEEE International Conference on Neural Information Processing, Perth, 1999, 435-440.

[108] M. Tang, K. Eshraghian, and H.N. Cheung, A hybrid approach to switchbox routing, *Proc. 26th International Conference on Computers and Industrial Engineering*, Melbourne, 1999, 235-241.

[109] A.Y. Tetelbaum, Generalized optimum path search, *IEEE Trans. on Computer-Aided Design*, Vol. 14, No. 12, 1995, 1586-1590.

[110] K.S. The, D.F. Wong, and J. Cong, A layout modification approach to via minimization, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 10, No. 4, 1991, 536-540.

[111] C.C. Tong and C.-L. Wu, Routing in a three-dimensional chip, *IEEE Transactions on Computers*, Vol. 44, No. 1, 1995, 106-117.

[112] G. Tong and C.L. Liu, Minimum crosstalk switchbox routing, *Integration, The VLSI Journal*, Vol. 19, No. 3, 1995, 161-180.

[113] J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Maryland, 1983.

[114] J.-S. Wang and R.C.T. Lee, An efficient channel algorithm to yield an optimal solution, *IEEE Trans. on Computers*, Vol. 39, No. 7, 1990, 957-962.

[115] C. Wiley, K.M. Lau, and S.A. Szygenda, m3D: a multidimensional dynamic configurable router, *Proc. IEEE International Symposium on Circuits and Systems*, 1993, 1857-1860.

[116] X.-M. Xiong and E.S. Kuh, The constrained via minimization for PCB and VLSI design, *Proc. 25th ACM/IEEE Design Automation Conference*, 1988, 573-578.

[117] X.-M. Xiong and E.S. Kuh, A unified approach to the via minimization problem, *IEEE Trans. on Circuits and Systems*, Vol. CAS-36, No. 2, 1989, 190-204.

[118] J.T. Yan and P.Y. Hsaio, CONVERGE: a circular-assignment-based switchbox router with via reduction, *IEE Proceedings Computers and Digital Techniques*, Vol. 42, No. 1, 1995, 72-76.

[119] T. Yoshimura and E.S. Kuh, Efficient algorithms for channel routing, *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol. CAD-1, 1982, 25-35.