On Planar Routing of Multi-Terminal Nets in VLSI Physical Design

Jahangir A. Hashmi

ON PLANAR ROUTING OF MULTI-TERMINAL NETS IN VLSI PHYSICAL DESIGN

by

Jahangir A. Hashmi

A Dissertation
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 1992

ON PLANAR ROUTING OF MULTI-TERMINAL NETS IN VLSI PHYSICAL DESIGN

Jahangir A. Hashmi, M.S.

Western Michigan University, 1992

In this thesis, we study two problems related to the physical design of VLSI circuits. One problem is related to the VLSI global routing and the other is related to the detailed routing in a bounded region.

We consider the routing of multiple multi-terminal nets on a single layer. We suggest a new approach for this problem. Our approach is based on simultaneously finding a forest of k non-intersecting spanning trees. We present an $O(n^3)$ algorithm for finding two spanning trees corresponding to two nets on a single layer.

We also study the problem of finding a minimum Steiner tree in the presence of obstacles when the terminals lie on the boundary of a rectilinear rectangle (RSTO) and present two results. Our first contribution is an exact solution for finding the shortest rectilinear Steiner tree in presence of an obstacle. Second, we give a provably good approximation algorithm for RSTO in the presence of n obstacles.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Naveed A. Sherwani, for his invaluable guidance, constant encouragement, and motivation. His continuous support made this work a reality.

I wish to thank the members of my thesis committee, Professor Alfred J. Boals and Professor Ajay K. Gupta, for their guidance and careful reading of the manuscript. I would like to thank Dr. J. Donald Nelson, the Computer Science Department Chair, for making all resources available "round the clock."

Thanks go to all the Night Group members for their continued support and for creating a nice research environment. Special thanks also go to my friends Hang Chen and Tim Strunk for their help in the proof of a theorem and implementation of an algorithm.

Jahangir A. Hashmi

Order Number 1347634

On planar routing of multi-terminal nets in VLSI physical design

Hashmi, Jahangir Alam, M.S.

Western Michigan University, 1992

To my parents.

TABLE OF CONTENTS

Table of Contents – continued

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

In the *Printed Circuit Boards* (PCBs) and *Integrated Circuits* (ICs) designs, the problem of routing wires through some interconnection medium, called the *Routing Problem*, is of critical importance. Due to the recent advances in the fabrication technology, the component density has increased manyfold. With the increased number of components and availability of a large number of routing layers, it is humanly not possible to route millions of wires in a reasonable time. Thus, the routing problem must be solved automatically. Most wire-routing problems are computationally hard: determining whether an instance of a routing problem is even solvable is usually NP-complete.

Routing problems abound, but they share some common characteristics. The wires, when routed, must connect certain points called *terminals* in a specified pattern, and they must satisfy some geometric constraints, such as, having a certain minimum thickness and separation from one another. Additional constraints may be imposed on the wires, e.g., that they be composed of rectilinear segments. The space in which wires are to be placed is called the *routing region.* In almost all practical problems, the routing region consists of one or more planes, or *layers*, with wires being allowed to pass between layers only at certain points.

The character of a routing problem depends largely upon the topology of the routing region. Multilayer routing problems are generally NP-complete [43] even when the routing region has a simple shape. For this reason, much of the theoretical work on multilayer routing has concentrated on approximation algorithms. [40]. These algorithms do not attempt to route within a fixed region, but instead they produce wirings that produce optimality in terms of the routing space or the number of layers they use. Single-layer routing problems are also

NP-complete in the general case [29, 39]. Several restricted single-layer routing problems are known to be efficiently solvable, however, including those in which the routing region is simply connected [12, 32, 41, 45] or annular [4] and the terminals lie on its boundary.

Routing problems are mainly of three kinds. In a pure routing problem, the routing region and the terminals are fixed; the algorithm must determine whether the wires can be routed, and if so, find feasible realizations (or detailed routings) for them. In most single-layer routing problems, one can also minimize the length of every wire, which is desirable from a practical standpoint. Sometimes one asks only whether the wires can be routed at all; then one is concerned with a *routability* problem. The NP-completeness results mentioned above apply to routability problems. In a *placement* problem, one thinks of the terminals as being attached to modules which can move. As modules move, the shape of the routing region may change. The issue is to find placements for the modules and feasible realizations for the wires so as to minimize some geometric quantity like the area of the routing region.

Recent advances in VLSI fabrication technology have made multiple routing layers available. Currently, up to 100 routing layer technology is available for Multiple-chip Modules (MCMs). Due to several practical reasons, it is desirable to route every net on a single routing layer. For example, the power bus, ground bus, clock signal nets, and some other critical nets should be entirely routed in one of the low-capacitance and low-resistance layers. Even for MCMs, although many routing layers are available, vias cause many problems and should be avoided. Thus it is desirable to find planar routing of nets.

With the availability of several routing layers and many practical aspects, planar (single layer) routing has gained a lot of attention. Several researchers have investigated the routability and the optimal routing of nets in the single layer. In [35] several single layer area routing algorithms are presented. Recently, an approximation algorithm has been presented which finds the planar routing of nets with the objective function of minimizing the number of layers.

Most of the single-layer routing studies were focused on the routability of two-terminal nets. In this thesis, we study two problems related to single-layer routing of multi-terminal nets: (1) routing of multiple multi-terminal nets in the plane, and (2) planar Switchbox routing in the presence of obstacles. Given $\mathcal{V} = \{V_1, V_2, \cdots, V_k\}$, $V_i = \{v_{i1}, v_{i2}, \cdots, v_{in_i}\}$ sets of terminal points in the plane, the k-STTP is the problem of finding k non-intersecting trees $T_1, T_2, \cdots, T_k$, where $T_i = (V_i, E_i)$ is a rectilinear Steiner tree over V_i. In this thesis, we study the k-STTP and suggest a totally new approach. Power and ground routing is a good example of 2-SPTP. Figure 1.1 shows an instance of power and ground routing and a solution produced by our algorithm. We also study the problem of finding minimum Steiner tree in presence of obstacles when the terminals lie on the boundary of the rectangle (RSTO) and present two results. Our first contribution is an exact solution for finding the shortest rectilinear Steiner tree in presence of an obstacle when the terminals lie on the boundary of the rectangle. This result extends the results of Cohoon et al. [9]. Secondly, we give an approximation algorithm for RSTO in presence of n obstacles. The bound of our algorithm is less than or equal to $\mu(S^*) + \sum_{i=1}^{n} \max\{L(O_i), W(O_i)\}$, where S^* is a Shortest Rectilinear Steiner Tree when no obstacle is present and $L(O_i)$ (resp. $W(O_i)$) represents the length (resp. width) of obstacle O_i. Figure 1.2 shows an instance of RSTO and a solution.

The rest of this thesis is organized in the following manner. We review the basic VLSI and graph theory terminology in the next chapter. In Chapter III, we consider the planar routing of two multi-terminal nets on the same routing layer. In Chapter IV, we study the switchbox routing problem. We conclude posing some open problems in Chapter V.

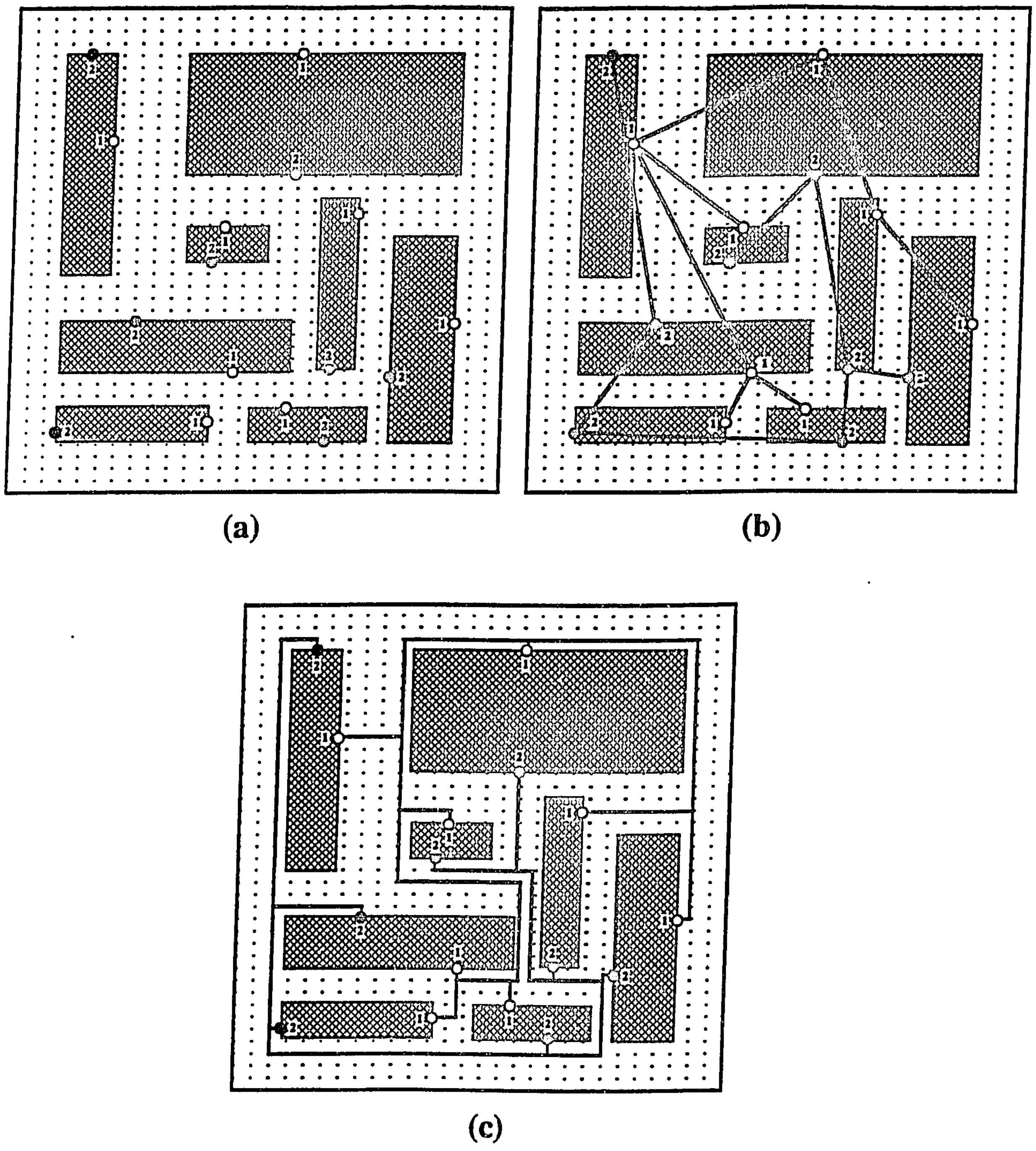


Figure 1.1 An Application of 2-Steiner Tree Problem.

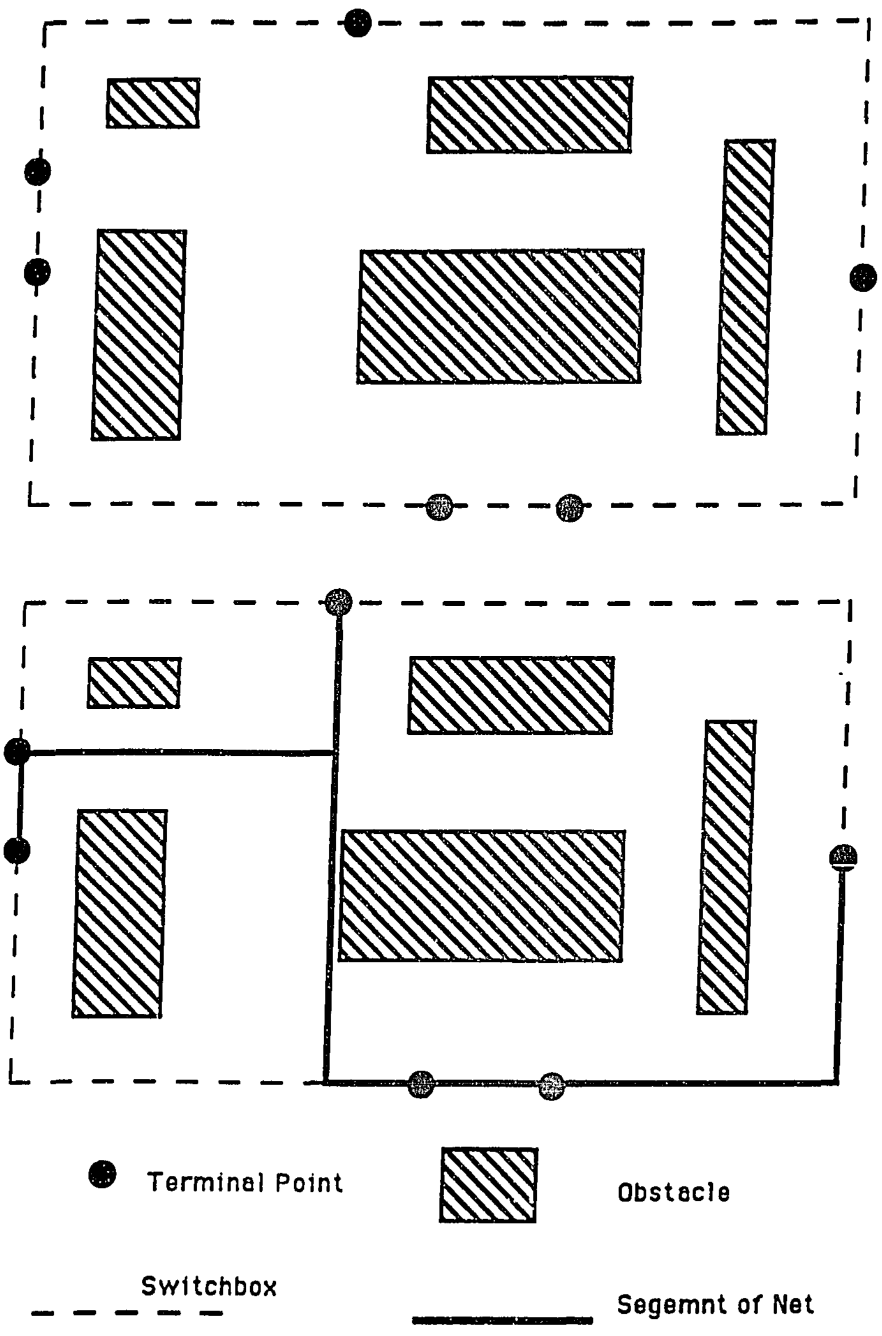


Figure 1.2 An Example of Switchbox Routing.

CHAPTER II

PRELIMINARIES

The exponential growth of the number of components per chip has increased the complexity of VLSI design problem manyfold. This has led the designers to automate a major part of the VLSI design cycle. In varying degrees, design automation is employed in all phases of design. An important area in which design automation has been quite successful is circuit layout. Circuit layout is an important part in the design process and highly prone to human errors. Designers therefore are highly motivated to make great strides in this area of design automation.

The circuit layout problem is broadly divided into two subproblems: *Placement* and *Routing.* By routing we mean the process of establishing desired connectivity among different terminals through conducting paths. The routing problems, in VLSI design, arise in many complex ways depending upon design methodology, level of integration and system goals.

The first phase of VLSI design cycle, called *system specification* and *functional design*, deals with the high level goals of the system along with the constraints, speed, area, etc. The output of the first phase is a functional description of the system that may be described in terms of a set of boolean expressions. The second phase is *logic design*, the purpose of which is to realize the set of boolean expressions generated in the first phase. One of the many available methodologies, *e.g.*, RAM, PLA or ROM may be used for this purpose. The third phase is called *circuit design.* In this phase the logic design obtained in phase two is mapped on to electronic circuits. The circuit design in fact depends on the type of chip design methodology being used by the designer. Currently, there are several approaches for chip design. Chip design methodologies include standard cell, sea of gates and gate-array, etc. Chips can also be custom designed. The output of this phase is a set of circuit blocks and their interconnections.

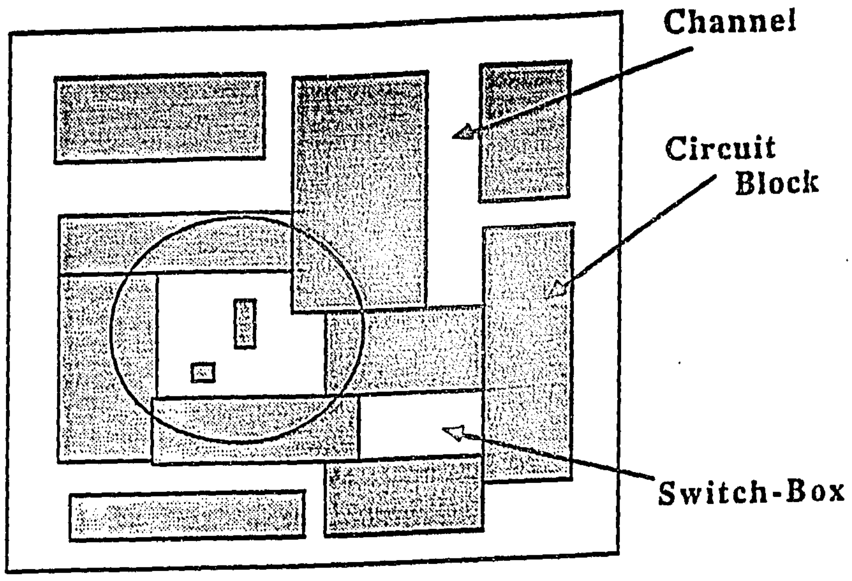
The next phase is called *circuit layout*, in this phase it is decided how the circuit blocks will be placed on the chip surface and how wires will be routed between two blocks that need to be connected. These two sub-phases are called placement and routing, respectively. The chip then goes through *design verification*, *testing* and *debugging* phases before it is committed to fabrication.

2.1 Routing Problems in VLSI

Routing is an important problem in circuit layout and between 25 to 50 percent of all VLSI layout design time is spent on routing. A set of terminals which are to be mutually connected is called a *net*. The purpose of routing is to establish connectivity among all terminals belonging to the same net, i.e., route wires between all terminals of a net. The space available for wiring is the space between two adjacent blocks. This space usually has rectilinear shape and may have some obstacles and internal boundaries as shown in Figure 2.1. The routing task is usually divided into three steps: (1) Channel and Switch-box identification, (2) Global Routing, and (3) Local Routing.

In the first step of routing a list of channels and switch-boxes available for routing is generated. The purpose of global routing is to assign wires to channels, without going into detailed routing. The usual approach is to route one net at a time sequentially until all nets are connected. To connect a net, a *greedy approach* or *maze runner* router is used. In most cases "rip-up and reroute" and manually routing is necessary to route those nets that can not be handled by global routers. This method is only useful if a reasonable order of nets can be found. Unfortunately no method exists for finding such an order.

Although global routing assigns nets to channels, the task of assigning nets to specific tracks within channel still remains. In the next step, called local



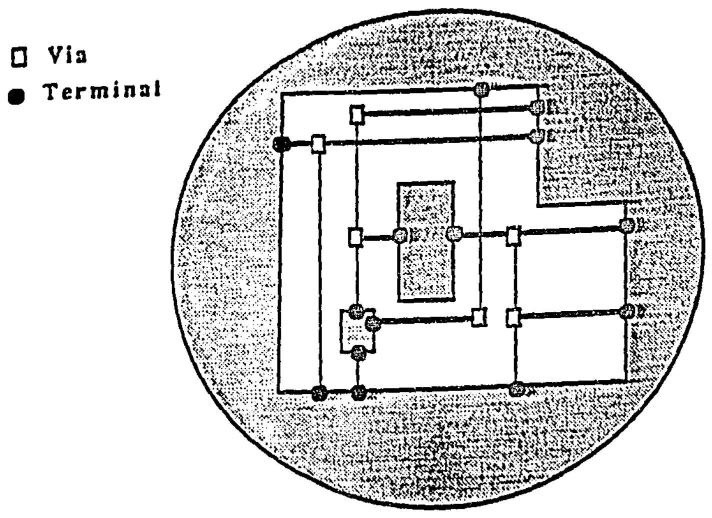


Figure 2.1 A Routing Problem.

routing, the nets are assigned to specific tracks, thus completing all details of routing.

In VLSI design, local routing is done by etching conductor paths on one or more *metallization* layers. The number of layers available depends on the fabrication technology. Traditionally, two layers have been used for routing. However, three metal layer technology is emerging. The routing may be done in a manner that each net consists of vertical and horizontal wire segments. Then, one layer is used for horizontal wire segments while the other layer is used for vertical wire segments. The electrical connection between two segments of a net is established through contact windows which are commonly known as *via-holes* or simply *vias*.

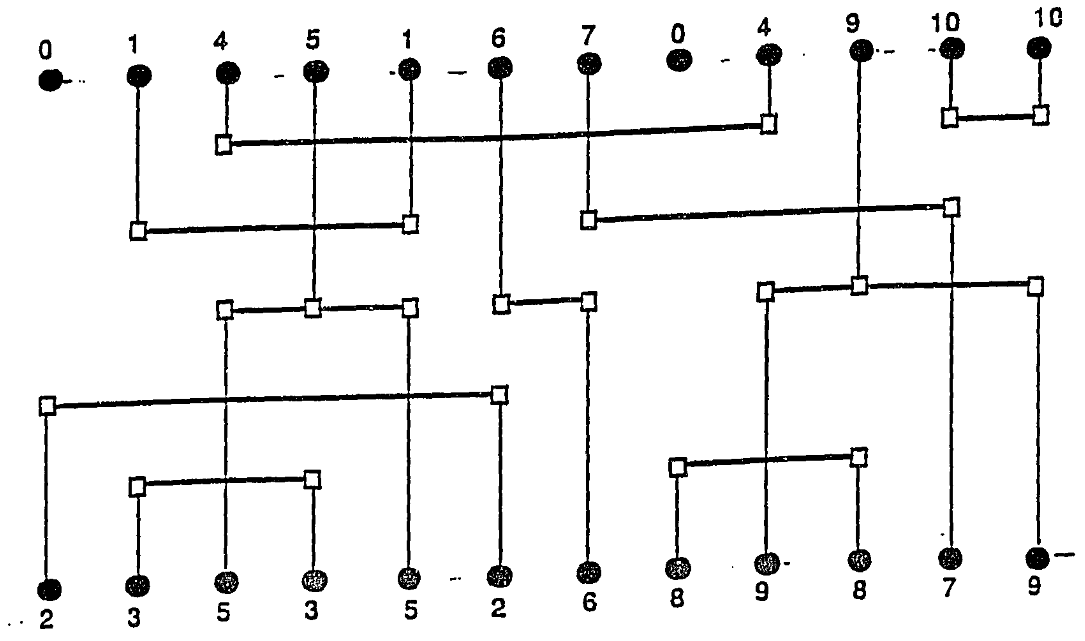
The local routing problem depends on the number of layers being used for routing. Thus based on number of layers, we have (a) Two layer and (b) One layer routing problems. Recently, however, three layer fabrication technology has been perfected so it is expected that in the future three and possibly multiple layer routing problems will also receive attention.

2.1.1 Two Layer Routing Problems

The routing problem is usually simplified by shape restrictions on the routing region. In theoretical studies the region is usually assumed to be rectangular and no obstacles are allowed. Thus depending on the shape constraints, the two layer routing problem is divided into two types of routing: *Channel Routing* and *Switch-box Routing.*

The channel routing problem can be stated as follows. A *channel* is a set of two horizontal rows of terminals at distance W units apart. The terminals on each side are numbered $0, 1, 2, \ldots, n$. The channel routing problem is then to connect all nets (set of terminals with same number) using two layers and to minimize the required channel width W. An example of channel routing is shown in Fig. 2.2.

One of the main advantages of channel routing is high packing density, i.e., more nets can be packed into smaller channel space. Therefore it has been



─	Wire segments in layer 1
━	Wire segments in layer 2
□	Via-hole or Via

Figure 2.2 An Example of Channel Routing.

used widely in automatic layout design methods. Several efficient channel routing algorithms exist, but these algorithms are heuristic in nature, since the channel routing problem with respect to several optimization criteria is known to be NP-Complete.

In the switch-box routing problem, in addition to two horizontal rows of terminals there are two vertical rows of terminals. In other words, if the routing region is closed instead of an open channel, then the problem is called switch-box routing.

2.1.2 Single Layer Routing Problems

If only one layer is available for routing, then some net configurations can not be routed. Therefore in case of single layer routing, feasibility of routing has to be checked first. Routability also depends on the routing model used. Depending on orientation of terminals the three main problems in single layer routing are: (1) River Routing, (2) Single Row Routing, and (3) Area Routing.

River routing is a channel routing problem restricted to a single layer. It is easy to verify using certain topological conditions whether routing is possible in one layer. Hsu [25] gave a simple algorithm to check feasibility and to do the actual routing if such a routing is possible. On the other hand, single row routing is a single layer routing problem in which terminals lie in a single line.

2.2 Graph Theory Terminology

A *graph* is a pair of sets $G = (V, E)$, where V is a set of vertices, and E is a set of pairs of distinct vertices called *edges*. If G is a graph, $V(G)$ and $E(G)$ are the vertex and edge sets of G, respectively. A vertex u is adjacent to a vertex v if $\{u, v\}$ is an edge, i.e, $\{u, v\} \in E$. The set of vertices adjacent to v is $Adj(v)$. An edge $e = \{v, u\}$ is *incident* with the vertices u and v, which are the ends of e. The degree of a vertex u is the number of edges incident with the vertex u.

Two graphs G and H are *isomorphic* if there is a bijection ϕ from $V(G)$ to $V(H)$ such that the vertices u and v are adjacent in G if and only if $\phi(u)$ and $\phi(v)$ are adjacent in H. A graph H is called the *complement* of graph $G = (V, E)$ if $H = (V, F)$, where, $F = V \times V - E$.

A graph H is a subgraph of a graph G if and only if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. If $E(H) = \{(u,v) \mid (u,v) \in E(G) \text{ and } u, v \subseteq V(H)\}$ then H is a *vertex induced subgraph* of G. Unless otherwise stated, by subgraph we mean vertex induced subgraph.

Let G be a graph. A *walk* is a sequence $P = v_0, e_1, \ldots, v_{k-1}, e_k, v_k$ of vertices v_i and edges e_j such that $e_i = \{v_{i-1}, v_i\}, 1 \leq i \leq k$. A *tour* is a walk in which all edges are distinct. A *path* is a tour in which all vertices are distinct. The *length* of a path(walk,tour) P given above is k. A path is a (u, v) path if $v_0 = u$ and $v_k = v$. A *cycle* is a tour of length $k, k > 2$ where $v_0 = v_k$ and $v_0, v_1, \ldots, v_{k-1}$ are distinct. A cycle is called *odd* if its length k is odd, otherwise it is an *even* cycle.

Two vertices u and v in G are *connected* if G has a (u, v) path. A graph is connected if all pairs of vertices are connected. A *connected component* of G is a maximal connected subgraph of G. An edge $e \in E(G)$ is called a *cut edge* in G if its removal from G increases the number of connected components of G by at least one. A *tree* is a connected graph with no cycles. A clique of a graph G is subgraph C such that $E(C) = V(C) \times V(C)$.

A *directed graph*, is a pair of set $(V, \vec{E})$, where V is a set of vertices and $\vec{E}$ is a set of ordered pairs of distinct vertices, called *directed edges* or *arcs*. We use the notation $\vec{G}$ for a directed graph. An arc $\vec{e} = \{u, v\}$ is incident with u and v, the vertices u and v are the *head* and *tail* of $\vec{e}$, respectively; $\vec{e}$ is an *in-arc* of v and an *out-arc* of u. The *in-degree* of u denoted by $d^-(u)$ is equal to the number of in-arcs of u, similarly the *out-degree* of u denoted by $d^+(u)$ is equal to the number of out-arcs of u. A directed graph $\vec{G} = (V, \vec{E})$ is called an *orientation* for a graph

$G = (V, E)$ by assigning an order to each edge. An orientation is called *transitive* if for each (u, v) and (v, w) there exists an edge (u, w).

Definitions of isomorphism, subgraph, path, walk are easily extended to directed graphs. A *directed acyclic graph* is a directed graph $\vec{G}$ with no cycles. A vertex u is an *ancestor* of v (and v is a *descendent* of u) if there is a (u, v) directed path in $\vec{G}$. A rooted tree (or *directed tree*) is a directed acyclic graph in which all vertices have in-degree 1 except one, the *root*, which has in-degree 0. The root of a rooted tree T is denoted *root(T)*. The *subtree* of tree T rooted at v is the subtree of T induced by the descendents of v. A *leaf* is a vertex in a directed acyclic graph with no descendents.

A *bipartite graph* is a graph G whose vertex set can be partitioned into two subsets X and Y, so that each edge has one end in X and one end in Y; such a partition (X, Y) is called *bipartition* of the graph. A *complete bipartite graph* is a bipartite graph with bipartition (X, Y) in which each vertex of X is adjacent to each vertex of Y; if $| X |= m$ and $| Y |= n$, such a graph is denoted by $K_{m,n}$. An important characterization of bipartite graphs is in terms of odd cycles. A graph is bipartite if and only if it contains no odd cycle.

Another interesting class of graphs based on the notion of cycle length is chordal graphs. If $C = v_0, e_1, \ldots, e_{k-1}, v_k$ is a cycle in G, a *chord* of C is an edge e in $E(G)$ connecting vertices v_i and v_j such that $e \neq e_i$ for any $i = 1, \ldots, k$. A graph is *chordal* if every cycle containing at least four vertices has a chord. Chordal graphs are also known as *triangulated graphs.*

A graph $G = (V, E)$ is a *comparability graph* if it is *transitively orientable.* A graph is called a *co-comparability graph* if the complement of G is transitively orientable.

A graph G is called an *interval* graph if and only if G is triangulated and the complement of G is a comparability graph. Interval graphs form a well known class of graphs and have been studied extensively [20]. Linear time algorithms are known for recognition, maximum clique, maximum independent set problems among others for this class of graphs [20]. The maximal cliques of an interval graph can be linearly ordered such that for every vertex $v \in V$, the cliques containing v occur consecutively [19]. Such an ordering of maximal cliques is called a *consecutive linear ordering.* An $O(\mid V \mid + \mid E \mid)$ algorithm for interval graph recognition that produces a consecutive linear ordering of maximal cliques is presented in [6].

A graph G is called a *permutation graph* if and only if G is a comparability graph and the complement of G is also a comparability graph. For the class of permutation graphs polynomial algorithms are known for recognition, maximum clique, maximum independent set and chromatic number [20, 37, 42].

Circle graphs, defined as the intersection graph of chords of a circle, can be recognized in polynomial time [13] and polynomial algorithms are also known for maximum clique and maximum independent set problems on circle graphs [17]. A polynomial algorithm is also known for the weighted maximum clique problem [25] but the chromatic number problem for circle graphs remains NP-complete [16].

The classes of graphs mentioned above are not unrelated; in fact, interval graphs and permutation graphs have a non-empty intersection. Similarly the classes of permutation and bipartite graphs have a non-empty intersection. On the other hand, the class of circle graphs properly contains the class of permutation graphs. In Figure 2.3 we show the relationship between these classes.

It is well known that the class of overlap graphs is equivalent to the class of circle graphs [20]. Similarly, the class of containment graphs is equivalent to the class of permutation graphs [20].

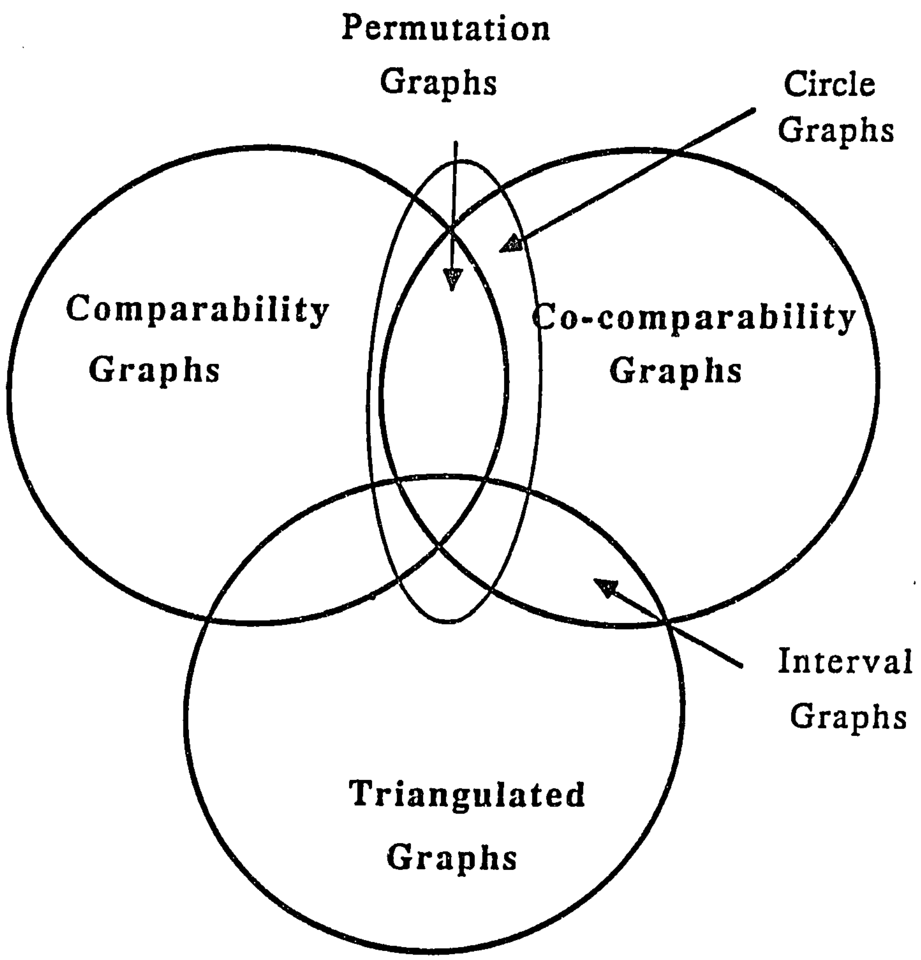


Figure 2.3 Relationship Between Different Classes of Graphs.

CHAPTER III

PLANAR ROUTING OF MULTIPLE MULTI-TERMINAL NETS

Given a set of terminal points N, the routing problem is to find a Steiner tree T over N, where $E(T)$ consists of only horizontal and vertical edges. Traditionally, a minimum spanning tree is used to approximate the cost and to find the initial topology, then that tree is rectilinearized to get the final solution. Given k sets of points in the plane, the problem is to find k non-intersecting trees corresponding to each of the k sets. The general approach followed for the multiple spanning tree problem is to first find the tree for a critical net and then to use heuristic approaches to find the trees for the other nets. We suggest a new approach for this problem. Our approach is based on simultaneously finding a forest of k non-intersecting spanning trees which span all the k sets, and rectilinearizing these trees one at a time. Thus, we are motivated to investigate the complexity of finding k non-intersecting spanning trees in the plane which span k sets of points.

Given a graph $G = (V, E)$, a *spanning tree* $T = (V, E_T), E_T \subseteq E$, is a connected graph without cycles. A *minimum cost spanning tree* (MST) of G is a spanning tree with minimum cost. According to a historical survey by Graham and Hell, the minimum spanning tree problem is most probably the first optimization problem. The MST problem is the easiest of all the network problems.

The minimum spanning trees have many applications. For example, in VLSI Physical Design, it is used in estimating wiring complexity and to approximate harder problems that are important in Layout, such as minimum Steiner tree problem.

There are several fast algorithms for finding the minimum cost spanning trees. In [44], Tarjan gave a survey of the minimum spanning tree problem, the first minimum cost spanning tree algorithm is attributed to a 1926 paper by

O. Boruvka. In 1956 Kruskal [30] gave the famous Kruskal's algorithm which can be implemented in $O(|E|\log|E|)$. Prim [38] invented the well known Prim's algorithm which finds a minimum cost spanning tree in $O(|E|\log|V|)$ time. All the above mentioned algorithms are greedy in nature.

Cheriton and Tarjan [8] present an algorithm that has $O(|E|\log\log|V|)$ run time. Gabow et al. [14] present the fastest algorithm known so far; it runs in time $O(|V|\log\beta(|E|,|V|))$, where

$$\beta(m,n) = \min\{i|\log^i(n) \leq \frac{m}{n}\},$$

$\log^i(n)$ denotes i-fold application of the base-2 logarithm. $\beta(m,n)$ is a very slowly growing function, its value does not exceed 5 as long as m,n are smaller than the number of elementary particles in the universe. This algorithm is difficult to implement and runs slowly on small examples.

The spanning trees of a special class of graphs, called Euclidean graph, induced by a set of points in the Euclidean space, are called *geometric spanning trees.* This chapter deals with the geometric spanning trees in the Euclidean plane. In this article, we use the terms spanning trees and minimum spanning trees to refer to geometric spanning trees and MSTs for the graphs in the Euclidean plane.

Let $V = \{V_1, V_2, \ldots, V_k\}$, $V_i = \{v_{i1}, v_{i2}, \ldots, v_{in_i}\}$ be the sets of points in the Euclidean plane. A *k-spanning tree* is a forest of trees $T_1, T_2, \ldots, T_k$, where T_i is a spanning tree over V_i, and all edges in the forest are non-intersecting. The *k-Spanning Tree Problem* (k-SPTP) is to find k-spanning trees which span over k sets of points in the plane without intersecting.

There are many practical applications of the multiple spanning trees. An interesting application exists in the *physical design* of VLSI systems. A set of terminal points (net) on a layer is to interconnected by the metal wires on the same layer. Given an assignment of k-nets to a layer, the problem is to find interconnections, embeddable in plane, such that no two wires belonging to different nets intersect. For a long time, MSTs have been used to find the approximate topology of the interconnection of a net. If the k-SPTP can be solved, then we can find the interconnection for k-nets. However, the k-STPT is significantly harder; thus, so far, only heuristic approaches have been used to find routing of more than one net.

To the best of our knowledge the k-spanning tree problem has not been considered to date. In this chapter, we show that the k-SPTP is computationally very hard, even the 2-SPTP is NP-complete. It is well known that 1-SPTP is optimally solvable in polynomial time. However, it is surprising that an instance of k-SPTP, $k \geq 2$, may not be solvable. This motivated us to investigate the necessary and sufficient conditions for the existence of a solution to a 2-SPTP. We give conditions under which the problem can not be solved. We present an efficient heuristic algorithm which runs in $O(n^3)$ time and produces solutions very close to the optimal, where $n = \max\{|V_1|, |V_2|\}$.

The rest of this chapter is organized as follows. In the next section we give the basic definitions and introduce the notation. We prove the NP-completeness of the k-SPTP in Section 3.3. In Section 3.4, we investigate the existence of a solution to a 2-SPTP, and give necessary and sufficient conditions. An efficient heuristic algorithm **Solve-2-SPTP** is presented in Section 3.5.

3.1 Preliminaries and Problem Formulation

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of points (vertices) in the Euclidean plane. An edge $e_{ij} = (v_i, v_j)$, also denoted as $e(v_i, v_j)$, is a line segment joining points v_i and v_j, $v_i, v_j \in V$. A graph $G = (V, E)$, is called a *complete graph over* V, denoted by $K(V)$, if $E = \{e(v_i, v_j) | \forall i, j \in \{1, 2, \cdots, |V|\}, i \neq j\}$. The *weight* of an edge, $\omega(e_{ij})$, is the Euclidean distance $d(v_i, v_j)$ between points v_i and v_j.

Let $G = (V, E)$ be a graph then, the *cost* of G, $\mu(G)$, is the total weight on all the edges in E, i.e.,

$$\mu(G) = \sum_{e_{ij} \in E} \omega(e_{ij}).$$

A vertex u is said to be connected to a vertex v in graph G if there exists a u-v path in G. A graph G is *connected* if every two of its vertices are connected. A graph that is not connected is *disconnected.* A *component* of G is a subgraph that is maximal with respect to the property of being connected. The number of components of G is denoted by $k(G)$.

A spanning tree of a graph $G = (V, E)$ is a connected graph $T = (V, E_T)$ without cycles. For a given set of vertices $V = \{v_1, v_2, \cdots, v_n\}$, there exists $2^{\binom{n}{2}}$ distinct spanning trees. A *minimum cost spanning tree* is a tree spanned over V with minimum cost. Let us formally define the minimum spanning tree.

Definition 1 Minimum Spanning Tree (MST): *Given a undirected graph $G = (V, E)$ with edge weighting function, $\mathcal{W} : E \rightarrow R^+$. Let $\mathcal{T} = \{T | T = (V, E_T)$ is a connected graph without cycles, $E_T \subseteq E\}$. Find a tree $T_{opt} \in \mathcal{T}$*

$$\mu(T_{opt}) = \min_{T \in \mathcal{T}} \sum_{e \in E(T)} \omega(e).$$

An *Euclidean graph* is a graph induced by a set of points in the Euclidean space. Spanning trees and MSTs of Euclidean graphs are called *geometric spanning trees* and *geometric MSTs* respectively.

The *convex-hull* $\mathcal{H}(V)$ is the smallest convex polygon which encloses all $v_i \in V$. A graph is called *planar* if all of its edges meet only at the vertices. The *crossing number* of a graph $G = (V, E)$ is defined as the number of edge crossings in the drawing of G in the plane. The *intersection number* of an edge $e_{ij} \in E$, $\psi(e_{ij})$, is the number of edges in E intersecting e. The crossing number of a plane graph is zero and the intersection number of all the edges is also zero. We use the notation $V(G)$ and $E(G)$ to represent the vertex set and the edge set of graph G respectively.

Definition 2 k-Spanning Tree Problem (k-SPTP): *Let $\mathcal{V} = \{V_1, V_2, \cdots, V_k\}$, $V_i = \{v_{i1}, v_{i2}, \cdots, v_{in_i}\}$ be the sets of terminal points in the plane. A k-spanning* *tree (k-SPT) is a forest of trees $T_1, T_2, \cdots, T_k$, where $T_i = (V_i, E_i)$ is a spanning tree over V_i, and all edges $e(v_{ij}, v_{ik}) \in E_i$ in the forest are non-intersecting, i.e., $\psi(e) = 0, \forall e \in E_i$, $1 \leq i \leq k$. Given $\mathcal{V}$, k-SPTP is to find a k-SPT.*

Let V_G be the set of all the points in the Cartesian plane. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a subset of V_G, then a *Steiner Tree* $\mathcal{ST}(V) = (V_r, E_r)$, is a tree which spans over V, where $E_r = \{e(v_i, v_j) | v_i, v_j \in V_r\}$ and $V \subseteq V_r \subset V_G$. The points in V are called *demand points*, and all the points in $(V_r - V)$ with induced degree of three or more in V_r are called *steiner points.* A *rectilinear Steiner tree*, denoted by $\mathcal{SR}(V) = (V, E_R)$ is a Steiner tree in which every edge $e_{ij} \in E_R$ is rectilinear, i.e., either vertical or horizontal. Let $\mathcal{S}$ be a set of all rectilinear Steiner trees which span over V, then a *Minimum Rectilinear Steiner Tree* (MRST), denoted by $SR^*(V)$, is a $\mathcal{ST}(V) \in \mathcal{S}$ with minimum cost.

Definition 3 **k-Steiner Tree Problem (k-STTP)**: *Let $\mathcal{V} = \{V_1, V_2, \cdots, V_k\}$, $V_i = \{v_{i1}, v_{i2}, \cdots, v_{in_i}\}$ be the sets of terminal points in the plane. A k-Steiner tree (k-STT) is a forest of trees $T_1, T_2, \cdots, T_k$, where $T_i = (V_i, E_i)$ is a rectilinear Steiner tree over V_i, and all edges $e(v_{ij}, v_{ik}) \in E_i$ in the forest are non-intersecting, i.e., $\psi(e) = 0, \forall e \in E_i$, $1 \leq i \leq k$. Given $\mathcal{V}$, k-STTP is to find a k-STT.*

3.2 2-SPTP is NP-Complete

In this section, we show that the k-SPTP is computationally very hard and even 2-SPTP is NP-Complete.

INSTANCE: Two non-empty finite sets of points in the Euclidean plane $V_1 = \{v_{11}, v_{12}, \cdots, v_{1n_1}\}$ and $V_2 = \{v_{21}, v_{22}, \cdots, v_{2n_2}\}$, and an integer $k \in \mathcal{Z}^+$.

QUESTION: Do there exist two spanning trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ which span over V_1 and V_2 respectively such that $\mu(T_1) + \mu(T_2) \leq k$, and no two edges in $E_1 \cup E_2$ intersect?

Theorem 1 *The 2-SPTP is NP-Complete and remains so even if no three points in $V_1 \cup V_2$ are collinear.*

Proof: We prove the NP-completeness by transforming the 2-independent sets problem to the 2-SPTP. We omit the details. ∎

3.3 Necessary and Sufficient Conditions for 2-SPTP

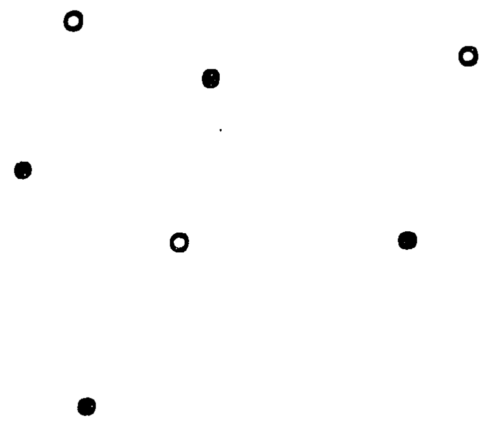
Given a set of vertices, a spanning tree always exists. But, given two sets of vertices V_1 and V_2 do there exist two non-intersecting spanning trees T_1 and T_2 which span the sets V_1 and V_2, respectively ? Figure 3.1 shows two simple 2-SPTP's. For the first example, shown in Figure 3.1(a), there exists a solution as shown in Figure 3.1(b). But for the second example, shown in Figure 3.1(c), there does not exist a solution. The above examples demonstrate the fact that there may or may not exist a solution to a 2-SPTP. This motivated us to investigate the necessary and sufficient conditions for the existence of a solution to a 2-SPTP. In this section we establish necessary and sufficient conditions for the existence of a solution to a 2-SPTP.

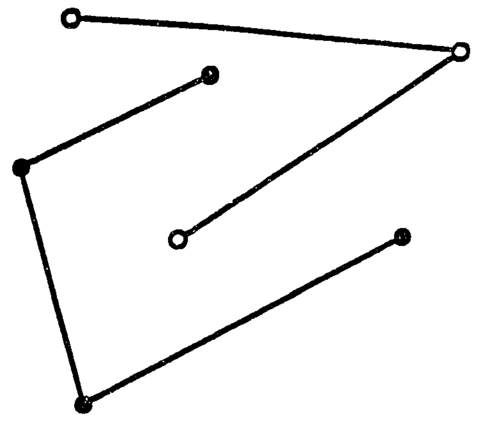
The convex-hull of a set of points encloses the region in which all the spanning trees exists. To establish the necessary and sufficient conditions for a solution to a 2-SPTP, we consider the relative positions of the convex-hulls of V_1 and V_2. Let $\mathcal{H}(V_1)$ and $\mathcal{H}(V_2)$ be the convex-hulls of V_1 and V_2, respectively. It is obvious that, for an instance of a 2-SPTP, if the two convex-hulls do not intersect then solution always exists. In fact, optimal solution can be found in $O(n \log n)$ time, where $n = \max\{|V_1|, |V_2|\}$, as stated in the following lemma.

Lemma 1 *If the convex-hulls $\mathcal{H}(V_1)$ and $\mathcal{H}(V_2)$ do not overlap then the 2-SPTP can be solved optimally.*

Now, we will define a term which will be used in establishing the conditions for the solution to the 2-SPTP.

Definition 4 Disconnecting Paths Pair (DPP): *A pair of vertices $v_{1x}, v_{1y} \in V_1$ is called a Disconnecting Paths Pair if every v_{1x}-v_{1y} path in $K(V_1)$ isolates a vertex $v_{2z} \in V_2$ or disconnects $K(V_2)$.*


(a) An instance of 2-SPTP.


(b) A solution to the 2-SPTP in (a).

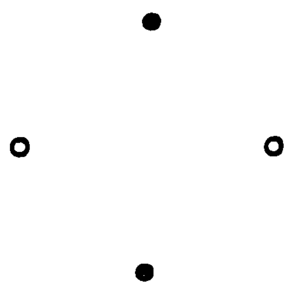

(c) An instance of 2-SPTP with no solution.

Figure 3.1 Two Examples of 2-SPTP.

Figure 3.2 shows a disconnecting paths pair in V_1 which isolates the vertex $v_{2x} \in V_2$. Note that disconnecting paths pairs always exists in pairs, i.e., if V_1 contains a DPP then V_2 also contains a DPP, similarly, if V_1 does not contain a DPP then V_2 would not contain a DPP. It seems quite obvious that, if there exists a DPP in V_1 or V_2 then the 2-SPTP can not be solved, as stated in the next theorem.

Theorem 2 *If there exists a disconnecting paths pair $v_{1x}, v_{1y} \in V_1$ then there does not exist a solution to the 2-SPTP.*

Proof: Let T_1 be the spanning tree over V_1, P be the unique path connecting u and v in T_1, let $w \in V_2$ be the vertex isolated by P. As w is isolated, there does not exist any tree T_2 which spans over V_2 without intersecting with T_1. All trees spanning over V_1 have a path connecting u and v, which isolates a vertex in V_2. Thus it would not be possible to find T_2 given any of those trees spanning over V_1. It implies that there does not exist any solution for this instance of the 2-SPTP. ■

Note that, if the convex-hull of one set is contained in the convex-hull of another set then there does not exist any DPP in $V_1 \cup V_2$. Now, we will show that if $\mathcal{H}(V_2)$ is contained in $\mathcal{H}(V_1)$ then 2-SPTP can be solved. Moreover, it can be solved in polynomial time. In order to show that, we first establish the following lemma.

Lemma 2 *If the convex-hull $\mathcal{H}(V_2)$ is contained in the convex-hull $\mathcal{H}(V_1)$,* i.e., *$\mathcal{H}(V_1) \supset \mathcal{H}(V_2)$, and there are no three collinear vertices in $V_1 \cup V_2$, then for every pair of adjacent vertices $v_{1i}, v_{1j} \in V_1$ there exist a vertex $v_{2x} \in V_2$, such that, the interiors of the triangle $v_{1i}v_{2x}v_{1j}$ and $\mathcal{H}(V_2)$ do not have any common part.*

Proof: Let $v_{11}, v_{12} \in V(\mathcal{H}(V_1))$ and let v_{11} be adjacent to v_{12} in the clockwise direction. Let U be the subset of $V(\mathcal{H}(V_2))$, such that, for any $u \in U$, the segment $\overline{v_{11}u}$ does not cross any edge in $E(\mathcal{H}(V_2))$. We select u_0 from U so that u_0 is the

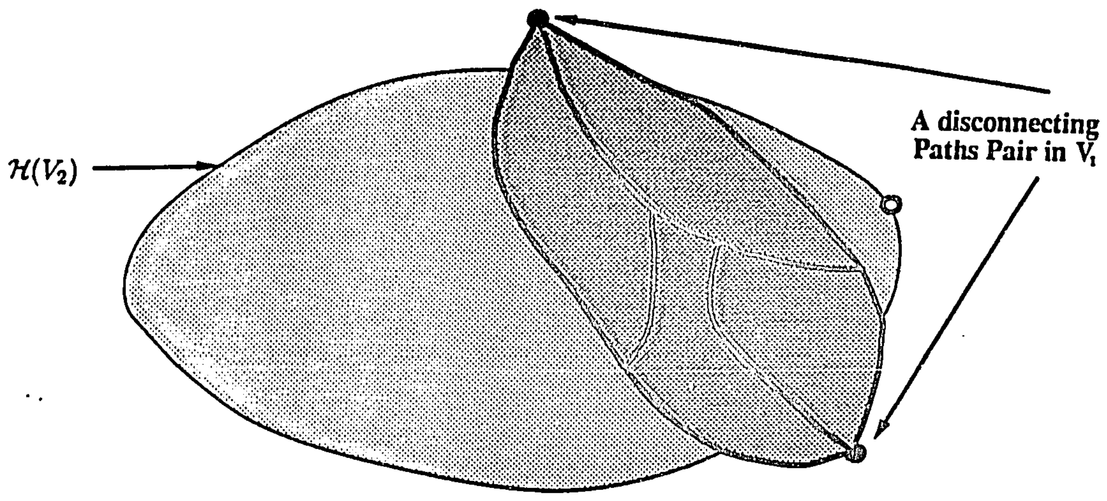


Figure 3.2 A Disconnecting Paths Pair.

last vertex in the clockwise direction. Now, we claim that the segment $\overline{v_{22}u_0}$ does not cross any edge in $E(\mathcal{H}(V_2))$. To show this, we observe that if $\overline{v_{12}u_0}$ crosses $\mathcal{H}(V_2)$, since no three points are collinear, then there exists at least one vertex $v_{2i} \in V_2$ such that u_0 is adjacent to v_{2i} clockwisely (see Figure 3.3). This contradicts to the selection of u_0. Therefore, the interior region of the triangle $v_{11}u_0v_{12}$ does not intersect with $\mathcal{H}(V_2)$. ■

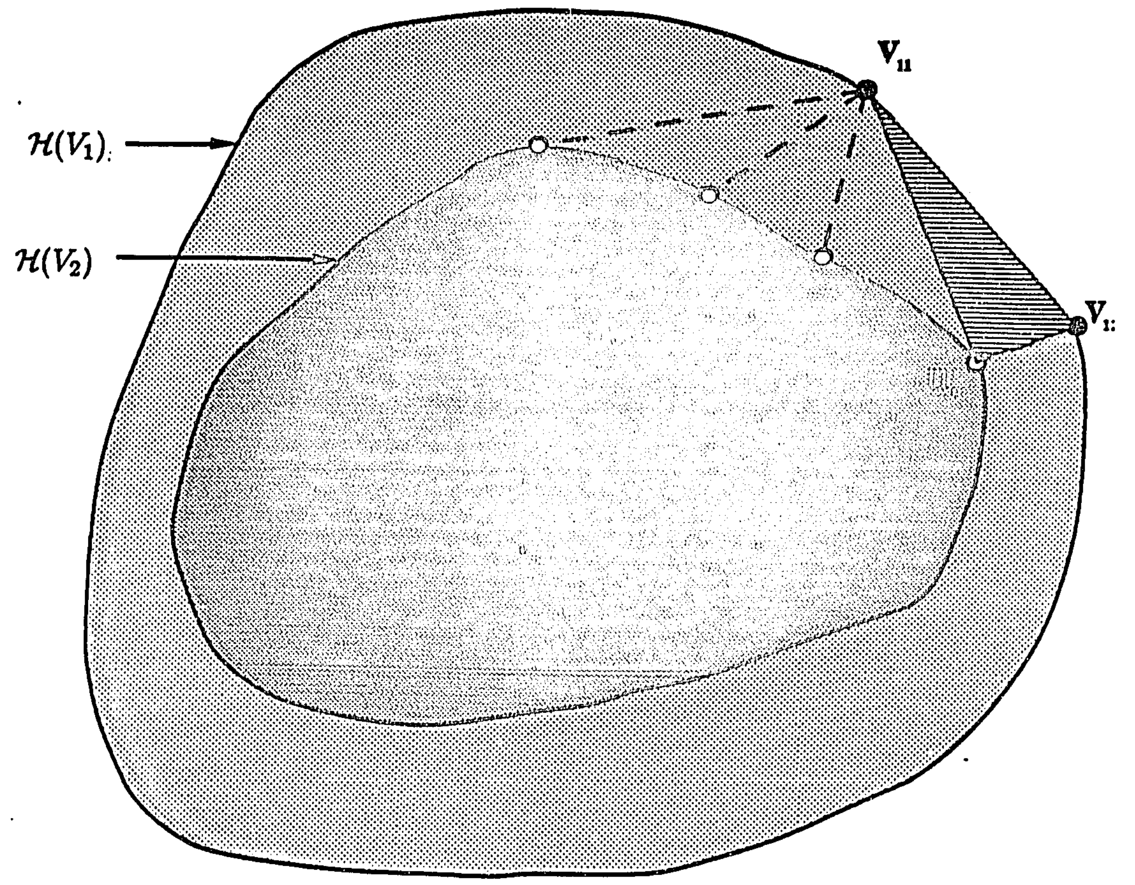
Based on Lemma 2, we now show that if one convex-hull is contained in another, then the 2-SPTP can be solved.

Theorem 3 *If the convex-hull $\mathcal{H}(V_2)$ is contained in the convex-hull $\mathcal{H}(V_1)$,* i.e., *$\mathcal{H}(V_1) \supset \mathcal{H}(V_2)$, and there are no three collinear vertices in $V_1 \cup V_2$ then there exists a solution to the 2-SPTP.*

Proof: We construct two sequences of convex-hulls, $\{\mathcal{H}(V_1^1), \mathcal{H}(V_1^2), \cdots, \mathcal{H}(V_1^n)\}$ and $\{\mathcal{H}(V_2^1), \mathcal{H}(V_2^2), \cdots, \mathcal{H}(V_2^m)\}$, where $V_1^1 = V_1$ and $V_2^1 = V_2$, so that $\mathcal{H}(V_1^1) \supset \mathcal{H}(V_2^1) \supset \mathcal{H}(V_1^2) \supset \mathcal{H}(V_2^2) \supset \cdots \supset \mathcal{H}(V_1^i) \supset \mathcal{H}(V_2^i) \supset \cdots$. Now, we will construct two spanning trees T_1 and T_2 which span over V_1 and V_2, respectively, and no two edges in $E(T_1) \cup E(T_2)$ intersect.

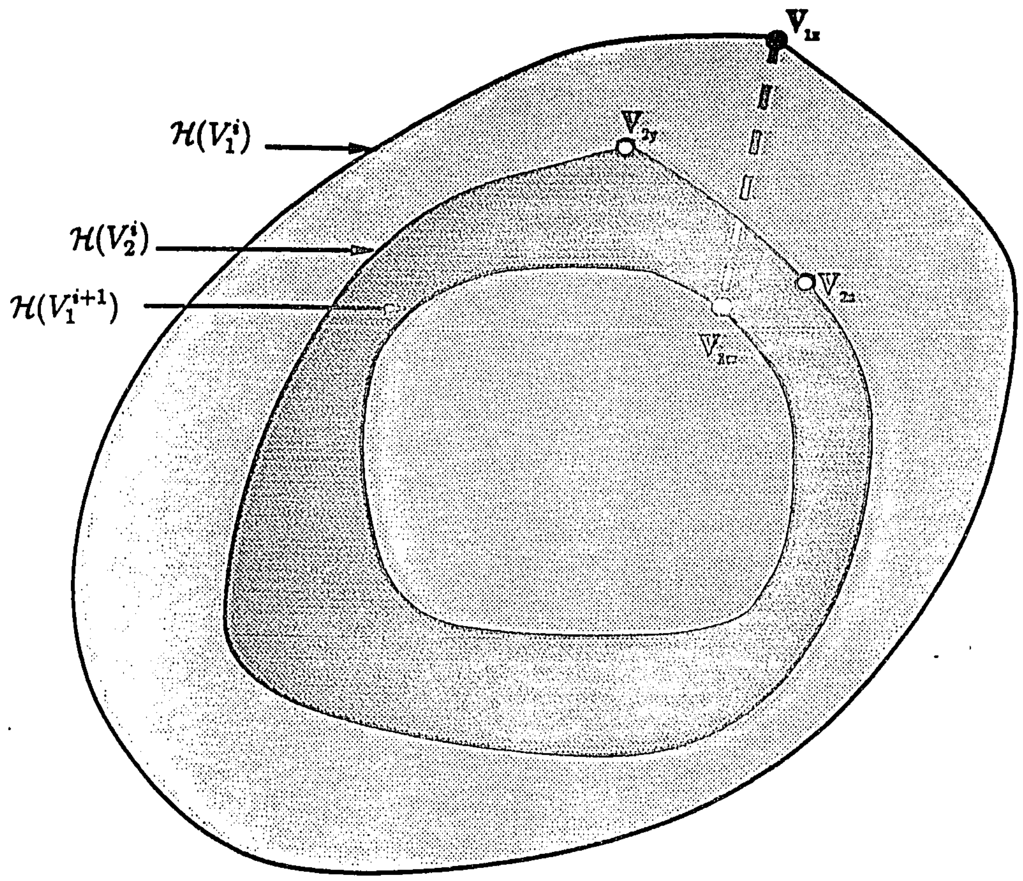
Since $\mathcal{H}(V_1^i) \supset \mathcal{H}(V_2^i) \supset \mathcal{H}(V_1^{i+1})$, by Lemma 2, for any point $v_{1x} \in V(\mathcal{H}(V_1^i))$ we can find two adjacent points $v_{2y}, v_{2z} \in V(\mathcal{H}(V_2^i))$ and a point $v_{1w} \in V(\mathcal{H}(V_1^{i+1}))$ so that the segment $\overline{v_{1x}v_{1w}}$ crosses $\overline{v_{2y}v_{2z}}$, see Figure 3.4. Now, we connect $\mathcal{H}(V_1^i)$ and $\mathcal{H}(V_1^{i+1})$ by connecting v_{1x} to v_{1w} and deleting the edge (v_{2y}, v_{2z}). Repeating the above process, we connect all the $\mathcal{H}(V_1^i)$s.

Since we used Lemma 2 to select a proper point for connecting $\mathcal{H}(V_1^i)$ and $\mathcal{H}(V_1^{i+1})$, every point in between $\mathcal{H}(V_1^i)$ and $\mathcal{H}(V_2^i)$ belongs to V_1, and it can be connected to one of the points of $V(\mathcal{H}(V_1^i))$, without crossing any edge $e \in E(\mathcal{H}(V_1^i)) \cup E(\mathcal{H}(V_2^i))$. In the same way, we connect all $\mathcal{H}(V_1^i)$s together. Now, we connect all the points in between $\mathcal{H}(V_1^i)$ and $\mathcal{H}(V_2^i)$ to points of $V(\mathcal{H}(V_1^i))$ and connect all points in between $\mathcal{H}(V_2^i)$ and $\mathcal{H}(V_1^{i+1})$ to points of $V(\mathcal{H}(V_2^i))$.



$\mathcal{H}(V_1) \supset \mathcal{H}(V_2)$

Figure 3.3 Construction for Lemma 2.



$\mathcal{H}(V_1^i) \supset \mathcal{H}(V_2^i) \supset \mathcal{H}(V_1^{i+1})$

Figure 3.4 One Convex Hull Contained in Another.

Finally, we delete one edge from the outer most convex-hull. If the inner most convex-hull is also a proper convex-hull then delete one edge from it. This final step ensures that we have two spanning trees which span over V_1 and V_2. ■

Now, we consider the case when two convex-hulls are overlapping. If the convex-hulls overlap then, there may or may not exist a disconnecting paths pair.

Theorem 4 *If all the vertices $v_{1j} \in V_1$ ($v_{2j} \in V_2$) in the region $\mathcal{H}(V_1) - \mathcal{H}(V_2)$ ($\mathcal{H}(V_2) - \mathcal{H}(V_1)$), can not be interconnected by a spanning tree T_1' (T_2') in the region enclosed by $\mathcal{H}(V_1) - \mathcal{H}(V_2)$ ($\mathcal{H}(V_2) - \mathcal{H}(V_1)$) then there does not exist a solution to the 2-SPTP.*

Proof: Let T_1' be a spanning tree which spans over all the vertices in the region enclosed by $\mathcal{H}(V_1) - \mathcal{H}(V_2)$ and T_2' be the tree which spans over all the vertices in the region enclosed by $\mathcal{H}(V_2) - \mathcal{H}(V_1)$. If the number of components of T_1' is greater than one then it is clear that if we connect these components, the connecting paths would have to go through the region enclosed by $\mathcal{H}(V_1) \cap \mathcal{H}(V_2)$. Therefore, at least one vertex $v \in V_2$ is isolated (see Figure 3.5). It is obvious that there exists at least one disconnecting paths pair. Thus according to Theorem 2 there does not exist a solution to this 2-SPTP. ■

Theorem 5 *If the convex-hulls $\mathcal{H}(V_1)$ and $\mathcal{H}(V_2)$ overlap, no three points in $V_1 \cup V_2$ are collinear, and there does not exist any disconnecting paths pair then there exist a solution to the 2-SPTP.*

Proof: We consider two regions $\mathcal{R}_1$ and $\mathcal{R}_2$ enclosed by $\mathcal{H}(V_1)$ and $\mathcal{H}(V_1) - \mathcal{H}(V_2)$, respectively.

Let us first consider the region $\mathcal{R}_1$. Let $V_2' \subseteq V_2$ be the set of points in V_2 contained in $\mathcal{R}_1$. Convex-hull $\mathcal{H}(V_2')$ is contained in $\mathcal{H}(V_1)$. Using an argument similar to the one used for Theorem 5, we can show that there exist two trees, T_1 and T_2', which span V_1 and V_2', respectively.

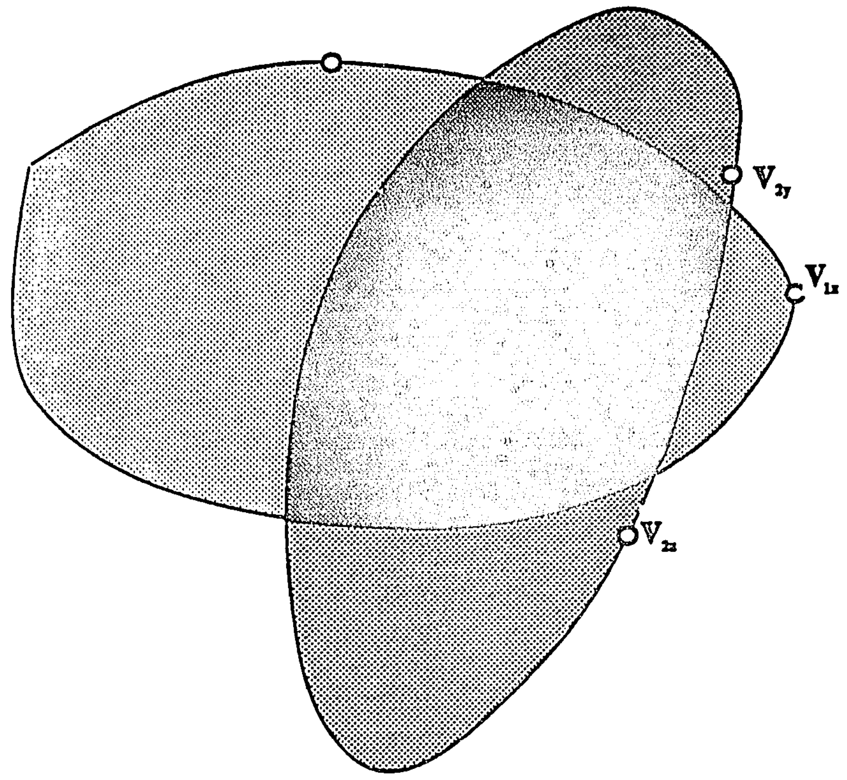


Figure 3.5 Intersecting Convex-hulls.

Next we consider the region $\mathcal{R}_2$. It is clear that $\mathcal{R}_2$ does not contain any points from V_1. Note that $\mathcal{R}_2$ may not be convex. Let $V_2'' = V_2 - V_2'$ be the set of points in region $\mathcal{R}_2$. We claim that there exists a spanning tree T_2'' which spans over V_2'' and T_2'' is contained in the region $\mathcal{R}_2$. To the contrary, assume that such a tree does not exist, then it contradicts with our assumption that there does not exist any DPP.

It remains to show that T_2' and T_2'' can be interconnected to form a tree T_2 which spans V_2 without disconnecting T_1. Let $\mathcal{H}(V_1^1)$ and $\mathcal{H}(V_2^1)$ be the outermost convex-hulls for V_1 and V_2', respectively. Consider any vertex $v_{2x}'' \in V_2''$, there exists a vertex $v_{2y} \in \mathcal{H}(V_2^1)$ such that the segment $\overline{v_{2x}''v_{2y}}$ intersects with exactly one edge of $E(\mathcal{H}(V_1^1))$. To show this, we use a geometric argument similar to the one used for Lemma 2. Thus $T_2 = T_2' + T_2'' + e(v_{2x}'', v_{2y})$ spans over V_2, but may disconnect T_1. Let $e_1 \in E(\mathcal{H}(V_1^1))$ be the edge intersecting with $e(v_{2x}, v_{2y})$, we remove e_1 from T_1, and add an edge e which was initially causing a cycle and was not included in T_1. ∎

Theorem 6 *If no three points in $V_1 \cup V_2$ are collinear, then there exists a solution to the 2-SPTP if and only if there does not exist any disconnecting paths pair.*

Proof: Follows directly from Theorems 2, 3, 4, and 5. ∎

Given a set of n points in the plane, it takes only $O(n \log n)$ time to the convex-hull. (Many faster algorithms also exist to find the convex-hulls). Thus the technique used in the proofs of this section, i.e., finding a sequence of convex-hulls, can be implemented to find a solution to a 2-SPTP in $O(n^2 \log n)$ time. However, this technique produces unacceptable results. Thus, we are motivated to find a better algorithm which can find solutions close to the optimal.

In this section we have given the necessary and sufficient conditions for the existence of solution to the 2-SPTP.

3.4 An Efficient Heuristic Solution for the 2-SPTP

In this section we present an efficient algorithm for the 2-SPTP problem. We first establish the novel theory behind our approach, which is followed by the algorithm and its detailed analysis.

3.4.1 Planar Graph Model

Euler discovered that, if G is a connected plane graph with n vertices, e edges, and r regions, then $n - e + r = 2$. This observation directly results in the following result.

Lemma 3 (Chartrand) *If G is a plane graph with n vertices, e edges and r regions, then*

$$n - e + r = 1 + k(G),$$

where $k(G)$ is the number of components of G.

Definition 5 Triangulated Planar Graph: *A planar graph G is called maximal planar, or triangulated planar graph if, for every pair of vertices u and v of G, the graph $G + e_{uv}$ is nonplanar. Thus for a maximal planar graph G, having order $n \geq 3$, the boundary of every region of G is a triangle.*

Theorem 7 (Chartrand) *If $G = (V, E)$ is a maximal planar graph with $|V| \geq 3$, then*

$$|E| = 3|V| - 6.$$

However, in this chapter, we consider only Euclidean edges, i.e., an edge between vertices u and v is a straight joining u and v. Thus, for a maximal planar graph with $|V| \geq 3$, $|E| \leq 3|V| - 6$. Therefore, for a maximum planar graph G, having order $n \geq 3$ the boundary of every region is a triangle, except the exterior region. Such a region enclosed by a triangle, is called a *triangulated region.*

Lemma 4 *Given a set of points V in the plane, a maximal planar graph can be found in $O(|V|^2 \log |V|^2)$ time.*

Proof: Given a set of points V, we first make a complete graph $G_1 = (V, E_1)$ over V, $O(|V|^2)$ time. Next we select $3|V| - 6$ non-intersecting edges from E_1 with minimum cost as first criteria. This selection process takes $O(|E_1| \log |E_1|) = O(|V|^2 \log |V|^2)$ time. ∎

3.4.2 The Algorithm

Algorithm **Solve-2-SPTP** finds a solution to the 2-SPTP if there exists one; otherwise, it declares failure.

```
Algorithm Solve-2-SPTP ($V_1, V_2$)
    (⋆ This algorithm takes two sets of vertices $V_1$ and $V_2$ and finds ⋆)
    (⋆ two non-intersecting trees $T_1$ and $T_2$ which span $V_1$ and $V_2$, respectively. ⋆)
  Input: Two sets of vertices $V_1 = \{v_{11}, v_{12}, \cdots, v_{1n_1}\}$ and
      $V_2 = \{v_{21}, v_{22}, \cdots, v_{2n_2}\}$.
  Output: Two spanning trees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$
      which span over $V_1$ and $V_2$.
begin
  1. (⋆ Find a triangulated planar graph over $V_1$. ⋆)
     $G_1 \leftarrow Triangulate(V_1)$
       (⋆ Let $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_r$ be the triangulated regions and
  2. (⋆ If all $v_2 \in V_2^{ext}$ in the exterior region don't form a spanning tree then⋆)
     (⋆ there does not exist a solution to 2-SPTP, quit. ⋆)
     $\tau_{ext} \leftarrow SpanTree(V_2^{ext}, G_1)$
     if $(k(\tau_{ext}) > 1)$ then
       solution does not exist, declare failure.
     end if
  3. (⋆ Make min. spanning trees $\tau_i$ over the set of vertices in each region $\mathcal{R}_i$ ⋆)
```

```
      for $i \leftarrow 1$ to $n$ do
        $\tau_i \leftarrow FindMST(K(V_2^i))$
      end do
   4. (⋆ Make a graph in which vertex corresponding to a set $V_2^i$, and edge ⋆)
      (⋆ being minimum cost edge between two $V_2^i$'s. ⋆)
      $V_R \leftarrow \emptyset, E_R \leftarrow \emptyset$
      for $i \leftarrow 1$ to $r$ do
        if $|V_2^i| > 0$ then $V_R \leftarrow V_R + v_i$
      for $i \in V_R$ do
        for $j \in V_R$ do
          if $i \neq j$ then
            $e_{ij} \leftarrow MinCostEdge(V_2^i, V_2^j)$,   $E_R \leftarrow E_R + \{e_{ij}\}$
   5. (⋆ Find a MST over $G_R$, remove edges in $E_1$ which intersect
          with any edge in $E_R$. ⋆)
      $T_R \leftarrow FindMST(G_R)$
      for $i \in E_1$ do
        if Intersects$(e_i, e_j)$, $\forall e_j \in E(T_R)$, then $E_1 \leftarrow E_1 - \{e_i\}$
      $T_I \leftarrow T_R \bigcup_{i=1}^{r} \tau_i$
   6. (⋆ Connect $\tau_{ext}$ to $T_I$ without disconnecting $G_1$. ⋆)
      (⋆ Remove all the excess edges from $G_1$ to form a tree $T_1$. ⋆)
      $X \leftarrow \{(u_2, v_2^i) | d(u_2, v_2^i)$ is minimum, $\forall u_2 \in V_2^{ext}$ and $\forall v_2^i \in V_2^i\}$
      $x \leftarrow MinCostEdge(X)$
      $T_2 \leftarrow T_I + \tau_{ext} + x$
      $T_1 \leftarrow FindMST(G_1)$

    return $(T_1, T_2)$
end Solve-2-SPTP
```

Many steps of the algorithm need to check the possible intersection of edges. Thus, we make two complete graphs $K(V_1)$ and $K(V_2)$. We make a new graph $G_X = (E(K(V_1)) \bigcup E(K(V_2)), E_X)$, where $E_X = \{e_{ij} | e_{1i} \in E(K(V_1)), e_{2j} \in E(K(V_2))$, and e_{1i} intersects with $e_{2j}\}$. Let $n = \max\{|V_1|, |V_2|\}$. Algorithm **Solve-2-SPTP** consists of 6 steps. We will describe each step in detail and analyze its complexity.

Triangulization

At Step 1, we find a triangulated planar graph $G_1 = (V_1, E_1)$ over V_1. From Lemma 4, we know that it takes only $O(|V_1|^2 \log |V_1|^2)$ time to find a triangulated planar graph G_1.

At the termination of Step 1, the convex-hull $\mathcal{H}(V_1)$ is divided into triangulated regions $\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_r$. The exterior region is denoted by $\mathcal{R}_{ext}$. We consider the set V_2 decomposed into subsets,

$$V_2 = V_2^{ext} \bigcup_{i=1}^{r} V_2^i,$$

where $V_2^i \subseteq V_2$ is a set of vertices in region $\mathcal{R}_i$ and $V_2^{ext} \subseteq V_2$ is the set of vertices in the exterior region $\mathcal{R}_{ext}$. In order to find each set V_2^i, we use a very simple method. For each vertex $v_{2j} \in V_2$, we find the region in which it lies. We make a line segment L which extends from the point $(x(v_{2j}), y(v_{2j}))$ to $(\infty, y(v_{2j}))$. Line segment L can intersect zero, one, or two edges of any triangulated region. If it intersects with exactly one edge of $\mathcal{R}_i$ then v_{2j} lies in $\mathcal{R}_i$, otherwise it does not.

```
    /\                    /\                    /\
   /  \                  /  \                  /  \
  /    \ o------->      / o--\------->       o-/----\---->
 /      \              /      \              /      \
--------              --------              --------
```

Necessary Conditions

At Step 2, the algorithm decides whether or not there exists a solution for the given instance of the 2-SPTP. This decision can be made if we can check the existence of a disconnecting paths pair. We check for the existence of a DPP in V_2. Note that, both the vertices of any disconnecting paths pair (v_{2x}, v_{2y}) in V_2 must be in the region $\mathcal{R}_{ext}$, i.e. $v_{2x}, v_{2y} \in V_2^{ext}$. Thus we check the visibility of the vertices in V_2^{ext}. If there exists a tree which spans V_2^{ext} without intersecting with any of the edges in E_1 then there does not exist a disconnecting paths pair; therefore, this instance of the problem is solvable according to Theorem. Otherwise, it is obvious that there does not exist a solution. In order to check this condition, we find a graph $G_2 = (V_2^{ext}, E_2)$, where $E_2 = \{e_{2i} | e_{2i} \in E(K(V_2^{ext}))$ and $e_{ji} \notin E_X, \forall e_{1j} \in E_1\}$, and E_1 is the set of edges of the triangulated planar graph over V_1. Now, there are two cases to consider: (1) $k(G_2) = \{0, 1\}$, and (2) $k(G_2) > 1$. If $k(G_2) = 0$ then $V_2^{ext} = \emptyset$, and if $k(G_2) = 1$ then we can find a tree τ_{ext} which spans over V_2^{ext} in $O(|V_2^{ext}| \log |V_2^{ext}|)$ time. It is clear that if $k(G_2) > 1$ then there exist at least one disconnecting paths pair. Thus, Step 2 takes $O(|V_2^{ext}| \cdot |E_1|) = O(n^3)$ time.

Local Minimum Spanning Trees

Step 3 is the simplest step of the algorithm. For each set $V_2^i, \forall i \in \{1, 2, \cdots, r\}$, we find a MST $\tau_i = (V_2^i, E_2^i)$ which spans over V_2^i. Note that, every edge $e_{2j} \in E_2^i$ is fully contained in region $\mathcal{R}_i$, i.e., it does not intersect with any edge in E_1. For any set V_2^i, such a MST can be obtained in $O(|V_2^i| \log |V_2^i|)$ time. Thus, Step 3 takes $O(r|V_2| \log |V_2|) = O(n^2 \log |V_2|)$ time, where $n = \max\{|V_1|, |V_2|\}$.

Global Connection of Local MST's

The next phase of the algorithm is to interconnect all τ_i's without disconnecting G_1. At Step 4, the algorithm makes a new complete graph $G_R = (V_R, E_R)$. Each vertex $v_i \in V_R$ corresponds to a set V_2^i if $|V_2^i| > 0$. Note that, if $|V_2^i| = 0$, then v_i does not exist. An edge $e_{ij} \in E_R$ corresponds to the edge $e(v_{2i}, v_{2j})$ with the minimum cost edge which connects a vertex from V_2^i to V_2^j. Formally, E_R is

$$E_R = \{e_{ij} | e_{ij} = e(v_{2k}, v_{2l}), \text{ and } d(v_{2k}, v_{2l}) \text{ is minimum }, \forall v_{2k} \in V_2^i, v_{2l} \in V_2^j\}.$$

Note that every edge in E_R intersects with at least one edge in E_1, but, we ignore this intersection for now. (It will be taken care of at the next step.) Given a complete graph $K(V_2)$ it takes $O(|V_2|)$ time to find an edge with minimum cost edge between two V_2^i's. There are $\frac{|V_R|\cdot(|V_R|-1)}{2}$ edges in E_R, where $|V_R| \leq |V_2|$. Thus, it takes $O(|V_2|^3)$ time to find G_R.

At Step 5, we make a tree T_I which spans over $V_2 - V_2^{ext}$. In order to make T_I, we remove some edges from G_1, but its connectivity is guaranteed. G_R contains all the minimum cost edges which can be used to interconnect all τ_i. Thus, we first find an MST T_R which spans G_R and remove all the edges in E_1 which intersect with edges in T_R. Note that, G_R is a graph in which each vertex represents a τ_i and an edge e_{ij} denotes the minimum cost edge from a vertex in V_2^i to a vertex in V_2^j. Thus, T_R provides a connectivity between τ_i's. Therefore, tree $T_I = T_R \bigcup_{i=1}^{r} \tau_i$ spans over $V_2 - V_2^{ext}$. It is easy to argue that G_1 does not get disconnected by removing the edges of G_1 intersecting with T_I; however, due to space constraint we omit the formal details.

Clean up phase

At Step 6, we connect T_I and τ_{ext}. We make a set of edges X in which each edge is the minimum cost edge from a point in a V_2^i to a point in V_2^{ext}. We select an edge $x \in X$ such that the removal of the edges of G_1 intersecting with x does not disconnect G_1. Note that, such an edge must exist, otherwise there exists a disconnecting paths pair, which is a contradiction. Thus, $T_2 = T_I + \tau_{ext} + x$ is a tree which spans V_2 without disconnecting G_1. There exists a tree T_1, an edge induced subgraph of G_1 which spans V_1 and does not intersect with T_2. In other words, T_1 is a MST obtained from G_1.

Set X contains at most r edges, where r is the number of triangulated regions, $r \leq |V_2|$. It takes $O(|V_2|^2)$ time to find X and select edge x. Finding a spanning tree T_1 from given graph G_1 takes $O(|V(G_1)| \log |V(G_1)|) = O(|V_1| \log |V_1|)$.

From the above discussion, we can derive the following theorem.

Table 3.1

Comparison of OPT-2 and Solve-2-SPTP

No.	$\lvert V_1 \rvert$	$\lvert V_2 \rvert$	$\mu(T_1)$	$\mu(T_2)$	$\mu(T_1) + \mu(T_2)$	μ(Opt. 2-SPT)
1.	5	10	56	112	168	146
2.	8	11	65	89	154	130
3.	15	15	72	65	137	137
4.	20	15	107	78	185	166
5.	20	20	92	120	212	212

Theorem 8 *Given an instance of a 2-SPTP, algorithm* Solve-2-SPTP *finds a solution in $O(n^3)$ time, if there exists one, otherwise it declares failure.*

3.5 Experimental Results

We have implemented algorithm Solve-2-SPTP in C on Sun Sparc station 1+. We have also developed a X-windows graphics user interface which provides an excellent development environment. An exponential time dynamic algorithm OPT-2 is also implemented. Algorithm OPT-2 finds an optimal solution to the 2-SPTP that is used to compare the solutions produced by Solve-2-SPTP. Figure 3.6 shows the step by step execution of algorithm Solve-2-SPTP for a randomly generated example.

We have tested our algorithms on several random and worst-case examples, see Table 3.1. The random examples were generated using normal distribution for $|V_1|$ and $|V_2|$. On average, algorithm Solve-2-SPTP produced results which are at most 5-10% more than the optimal.

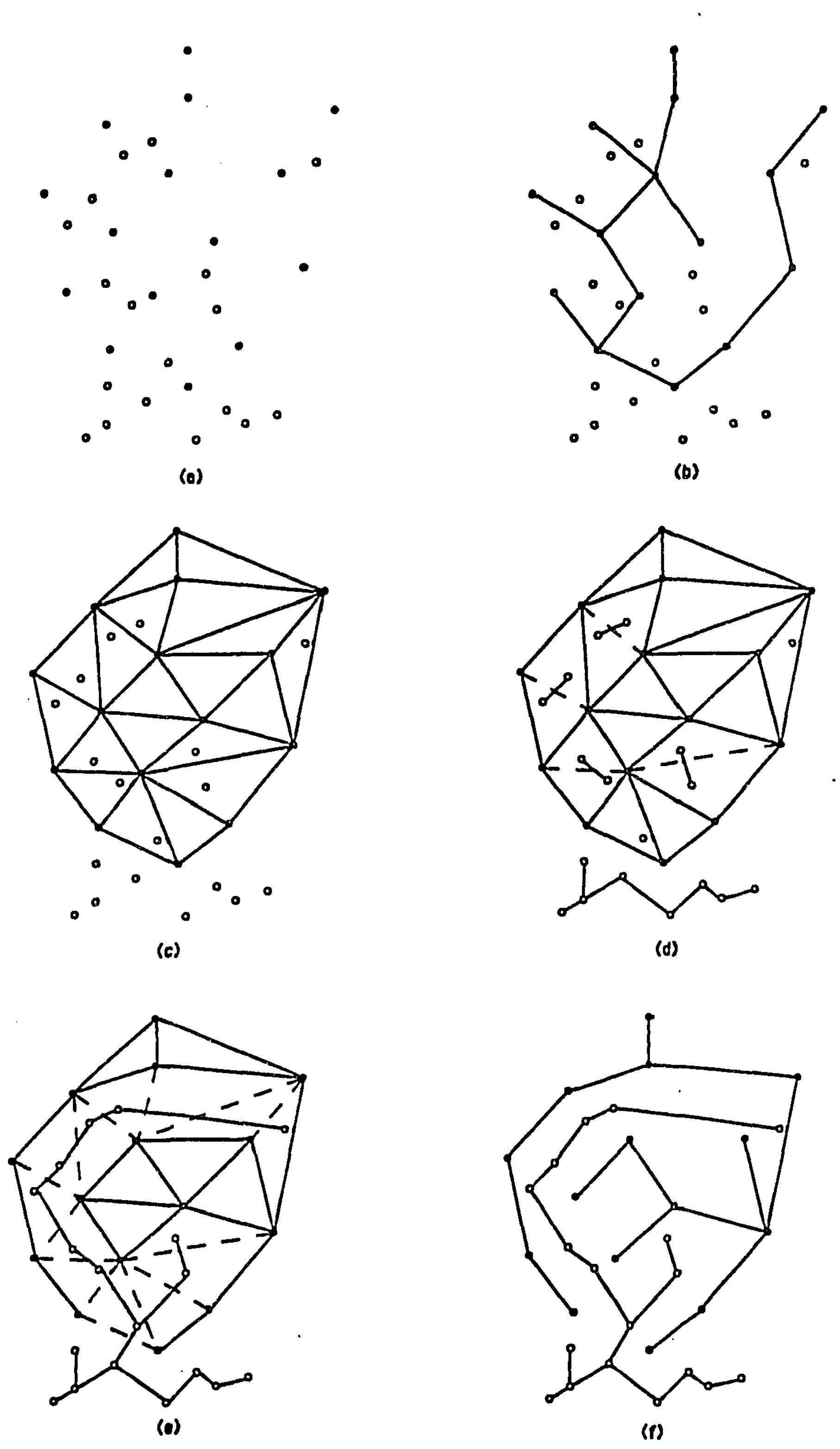


Figure 3.6 Step by Step Execution of Algorithm Solve-2-SPTP.

CHAPTER IV

PLANAR SWITCHBOX ROUTING IN THE PRESENCE OF OBSTACLES

Rectilinear Steiner trees play an important role in global routing of $VLSI$ circuits and consequently have been studied extensively [15, 9, 1, 22, 23, 46, 27]. Given a set S of specified vertices (*demand points*) and some arbitrary points (*Steiner points*) , a tree interconnecting these points is known as *Steiner tree* or *geometric Steiner tree* of set S. The Steiner tree with minimum total length is known as *Shortest Steiner Tree.* The *rectilinear Steiner tree* (RST) is a Steiner tree with only rectilinear edges.

The minimum cost rectilinear Steiner tree problem is known to be NP-complete [15]. Hwang in [27] showed that the ratio of the cost of the rectilinear minimum spanning tree (RMST) to that of an optimal RST is less than or equal to 3/2. Bern [5] gave an improvement to the dynamic programming algorithms that compute optimal Steiner trees spanning vertices in planner networks. As the general RST problem is NP-hard, many special case problems were studied and several heuristics have been developed. In [2] Aho et al. have considered two special cases: (1) Points lying on two horizontal lines, and (2) Points lying on the boundary of a rectilinear rectangle, and they gave a dynamic programming algorithm for constructing minimum RST in $O(n)$ in the first case and $O(n^3)$ time for the second case. Cohoon et al. [9] and Agarwal et al. [1] have presented linear-time exact algorithms for finding the shortest Steiner tree for points lying on the boundary of the rectangle.

The main focus of this chapter is routing using Steiner trees in presence of obstacles, which to best of our knowledge has not been considered before. However, the problem of rectilinear shortest paths in the presence of rectangular obstacles has been studied and developed an $O(n \log n)$ algorithm was presented

[11]. Joseph Jájá and Alice Wu developed a polynomial time algorithm to determine whether the given set of two terminal nets is routable in two-layer or knock-knee model for any fixed size sets in presence of obstacles [28].

In this chapter, we consider the problem of finding minimum Steiner tree in presence of obstacles when the terminals lie on the boundary of the rectangle (RSTO) and present two results. Our first contribution is an exact solution for finding the shortest rectilinear Steiner tree in presence of an obstacle when the terminals lie on the boundary of the rectangle. This result extends the results of Cohoon et al. [9]. Secondly, we give an approximation algorithm for RSTO in presence of n obstacles. The bound of our algorithm is less than or equal to $\mu(S^*) + \sum_{i=1}^{n} \max\{L(O_i), W(O_i)\}$, where S^* is a Shortest Rectilinear Steiner Tree when no obstacle in present and $L(O_i)$ ($W(O_i)$) represent the length (width) of obstacle O_i. To the best of our knowledge, the results reported in this chapter are the first to address the problem of obstacles in a Steiner tree setting.

The rest of the chapter is organized as follows: In the next section we formalize the problem and give necessary definitions. In Section 4.3, we consider 1-RSTO and give bounds for S_1^*. In Section 4.4, we present an efficient algorithm which produces very good results.

4.1 Preliminaries and Problem Formulation

Let $\mathcal{P}_1 = (x_1, y_1)$ and $\mathcal{P}_2 = (x_2, y_2)$ be two points in the Cartesian plane, (in this chapter we assume that for all points $x_i \geq 0, y_i \geq 0$) where x_i and y_i are the x- and y-coordinates of point $\mathcal{P}_i$, respectively. The functions $x(\mathcal{P}_i)$ and $y(\mathcal{P}_i)$ give the x- and y-coordinates of $\mathcal{P}_i$. The *horizontal distance*, *vertical distance* and the *rectilinear distance* between two points $\mathcal{P}_1$ and $\mathcal{P}_2$ are given by:

$$\begin{aligned}
\textit{Horizontal distance}: \quad h(\mathcal{P}_1, \mathcal{P}_2) &= |x(\mathcal{P}_1) - x(\mathcal{P}_2)| \\
\textit{Vertical distance}: \quad v(\mathcal{P}_1, \mathcal{P}_2) &= |y(\mathcal{P}_1) - y(\mathcal{P}_2)| \\
\textit{Rectilinear distance}: \quad d(\mathcal{P}_1, \mathcal{P}_2) &= h(\mathcal{P}_1, \mathcal{P}_2) + v(\mathcal{P}_1, \mathcal{P}_2).
\end{aligned}$$

If $\mathcal{P}_i$ and $\mathcal{P}_j$ are collinear, i.e., either $y(\mathcal{P}_1) = y(\mathcal{P}_2)$ or $x(\mathcal{P}_1) = x(\mathcal{P}_2)$, then an edge $e_{ij} = (\mathcal{P}_i, \mathcal{P}_j)$, is a line segment from $\mathcal{P}_i$ to $\mathcal{P}_j$, and e_{ij} is said to be incident on points (vertices) $\mathcal{P}_i$ and $\mathcal{P}_j$. If $y(\mathcal{P}_1) = y(\mathcal{P}_2)$ then $e_{ij} = (\mathcal{P}_i, \mathcal{P}_j)$ is called a *horizontal edge*, and if $x(\mathcal{P}_1) = x(\mathcal{P}_2)$ then it is called a *vertical edge.* The *weight* of an edge $e_{ij} = (\mathcal{P}_i, \mathcal{P}_j)$, $\psi(e_{ij})$, is the rectilinear distance between $\mathcal{P}_i$ and $\mathcal{P}_j$, i.e., $\psi(e_{ij}) = d(\mathcal{P}_i, \mathcal{P}_j)$. Two edges e_{ij} and e_{kl} are called *adjacent* if they are incident on a point, i.e., if either $i = k, i = l, j = k$, or $j = l$. A *rectilinear path* $P(\mathcal{P}_i, \mathcal{P}_j)$ is a sequence of adjacent non-intersecting edges from $\mathcal{P}_i$ to $\mathcal{P}_j$, and the *length* of rectilinear path $P(\mathcal{P}_i, \mathcal{P}_j)$, denoted as $\lambda(P(\mathcal{P}_i, \mathcal{P}_j))$, is the summation of the weights of all the edges in path P. We define a *complete vertical edge* to be a maximum sequence of adjacent vertical edges. Similarly, we define *complete horizontal edge* to be a maximum sequence of adjacent horizontal edges.

Let V_G be the set of all the points in the Cartesian plane. Let $V = \{v_1, v_2, \ldots, v_n\}$ be a subset of V_G, then a *Rectilinear Steiner Tree* (RST), $S = (V_r, E_r)$, is a tree which spans over V, where $E_r = \{e_{ij} = (v_i, v_j) | v_i, v_j \in V_r\}$ and $V \subseteq V_r \subset V_G$. The points in V are called *demand points*, and all the points in $(V_r - V)$ with induced degree of three or more in S are called *steiner points.* The *cost* of a *RST* $S = (V_r, E_r)$, denoted as $\mu(S)$, is the summation of the weights of all the edges in E_r, i.e., $\mu(S) = \sum_{e_{ij} \in E_r} \psi(e_{ij})$. Let $\mathcal{S} = \{S | S \text{ spans over } V\}$ be a set of *RST*s which span over V, then a *Shortest Rectilinear Steiner Tree* (SRST), $S^* = (V_s, E_s)$, is a *RST* in $\mathcal{S}$ with minimum cost.

Let $\mathcal{R} = (\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4)$ be a rectilinear rectangle in the Cartesian plane, where $\mathcal{P}_1 = (x_1, y_1)$, $\mathcal{P}_2 = (x_2, y_1)$, $\mathcal{P}_3 = (x_1, y_2)$, $\mathcal{P}_4 = (x_2, y_2)$, $x_1 < x_2$ and $y_1 < y_2$. We refer to $\mathcal{P}_1$ as the *bottom-left-corner* of $\mathcal{R}$, also given by the function $bl(\mathcal{R})$. Similarly, we refer to $\mathcal{P}_2 = br(\mathcal{R})$, $\mathcal{P}_3 = tr(\mathcal{R})$, and $\mathcal{P}_4 = tl(\mathcal{R})$ as *bottom-right-*, *top-left-*, and *top-right-* corners of $\mathcal{R}$. We define the length $L(\mathcal{R})$ of $\mathcal{R}$ as $L(\mathcal{R}) = d(tl(\mathcal{R}), tr(\mathcal{R})) = d(bl(\mathcal{R}), br(\mathcal{R}))$ and the width $W(\mathcal{R})$ of $\mathcal{R}$ as $W(\mathcal{R}) = d(tl(\mathcal{R}), bl(\mathcal{R})) = d(tr(\mathcal{R}), bl(\mathcal{R}))$. We call the edges $B_t = (\mathcal{P}_3, \mathcal{P}_4) = t(\mathcal{R}), B_b =$

$(\mathcal{P}_1, \mathcal{P}_2) = b(\mathcal{R}), B_l = (\mathcal{P}_1, \mathcal{P}_3) = l(\mathcal{R})$, and $B_r = (\mathcal{P}_2, \mathcal{P}_4) = r(\mathcal{R})$ the *top-*, *bottom-*, *left-*, and *right-*boundaries of $\mathcal{R}$, respectively.

Let $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$ be a set of non-intersecting rectangles, where each O_i is properly contained in $\mathcal{R}$, i.e., $x_1 < x(tr(O_i)) < x_2$, $y_1 < y(tr(O_i)) < y_2$, $x_1 < x(tl(O_i)) < x_2$, $y_1 < y(tl(O_i)) < y_2$, $x_1 < x(br(O_i)) < x_2$, $y_1 < y(br(O_i)) < y_2$, and $x_1 < x(bl(O_i)) < x_2$, $y_1 < y(bl(O_i)) < y_2$. The corners and boundaries for obstacle O_i are defined analogously to those of $\mathcal{R}$.

We define $V_t = \{t_1, t_2, \ldots, t_{n_t}\}$, $V_t \subset V_G$, $x_1 \leq x(t_i) \leq x_2$, $y(t_i) = y_2$, $x(t_i) < x(t_{i+1})$, to be the set of all the demand points on the top boundaries of $\mathcal{R}$. Analogously, we define $V_b = \{b_1, b_2, \ldots, b_{n_b}\}$, $V_l = \{l_1, l_2, \ldots, l_{n_l}\}$, and $V_r = \{r_1, r_2, \ldots, r_{n_r}\}$ to be the set of demand points on the bottom, left, and right boundary of $\mathcal{R}$, respectively.

Let $V' = V_t \bigcup V_b \bigcup V_l \bigcup V_r$ be the set of all the demand points. Let S^* be the SRST over V' when there is no obstacle inside $\mathcal{R}$. Let S_k be a *RST* and S_k^* be the *SRST* which spans over V' in the presence of k obstacles without intersecting with any obstacles. If for an edge e_{ij}, $e_{ij} - B_t - B_b - B_r - B_l \neq \emptyset$ then it is called an *exterior edge*, and is called *interior edge* otherwise. A *complete interior edge* is a complete edge which is adjacent to points on opposite sides of $\mathcal{R}$.

Let the *left set* of a point $\mathcal{P}_i$ be $LS(\mathcal{P}_i) = \{Y | x(Y) < x(\mathcal{P}_i), y(Y_b) \leq y(\mathcal{P}_i) \leq y(Y_t)\}$, where Y is either $r(O^i)$, $r(\mathcal{R})$, or a vertical edge in the current solution, Y_b and Y_t are the points on which Y is incident. We define the *left projection* from $\mathcal{P}_i$, $\rho_l(\mathcal{P}_i) = (\mathcal{P}_i, \mathcal{P}_j)$, to be an edge from $\mathcal{P}_i$ to a point $\mathcal{P}_j$ on $Y \in LS(\mathcal{P}_i)$ closest to $\mathcal{P}_i$, $\mathcal{P}_j$ is called the *left neighbor* of $\mathcal{P}_i$. Similarly, we define *right projection* $\rho_r(\mathcal{P}_i)$, *top projection* $\rho_t(\mathcal{P}_i)$ and *bottom projection* $\rho_b(\mathcal{P}_i)$, of a point $\mathcal{P}_i$.

4.2 SRST in the Presence of One Obstacle

Cohoon et al. in [9] and Agarwal et al. in [1] have independently given linear-time algorithms for finding optimal Steiner tree when all the terminal points lie on the boundary of a rectilinear rectangle. They have also shown that the interior of an SRST S^* could only have the topologies shown in Figure 4.1. In this section, we extend their work and consider the same problem in the presence of an obstacle O_1 inside the rectangle $\mathcal{R}$. We show that, there exists a RST S_1 in the presence of O_1 such that $\mu(S_1) \leq \mu(S^*) + \max\{L(O_1), W(O_1)\}$. We also give an exact algorithm for finding SRST S_1^*.

To show the bound, we consider all the topologies and all the possible locations of O_1 inside $\mathcal{R}$. We show that, whenever O_1 obstructs S^* then there exists a RST S_1 such that $\mu(S_1) \leq \mu(S^*) + \max\{L(O_1), W(O_1)\}$.

Theorem 9 *Let V be a set of vertices on the boundary of $\mathcal{R}$ and S^* be a SRST when no obstacle is present inside the switchbox. Then if an obstacle O_1 is introduced in the switchbox, S^* could be modified to a SRT S_1 such that $\mu(S_1) \leq \mu(S^*) + \max\{L(O_1), W(O_1)\}$ and any of the edges of S_1 do not intersect with the obstacle.*

Proof: In order to show the bound, we will break up the problem and consider the possible location of the obstacle with respect to S^* and the construction of S_1, within the bound. There are five cases to consider. Without loss of generality, we assume that $W(O_1) \geq L(O_1)$.

Case 1 - *Obstacle intersects with a cross vertex:* Consider the cross vertex and the obstacle shown in Figure 4.2(a). Without loss of generality, assume that $a \leq b$ and $c \leq d$. Figure 4.2(b) shows the modified tree S_1 which is very similar to S^* except that the portion intersecting with O_1 is deleted and few additional edges are added around the boundary of the obstacle. It is clear that the total added

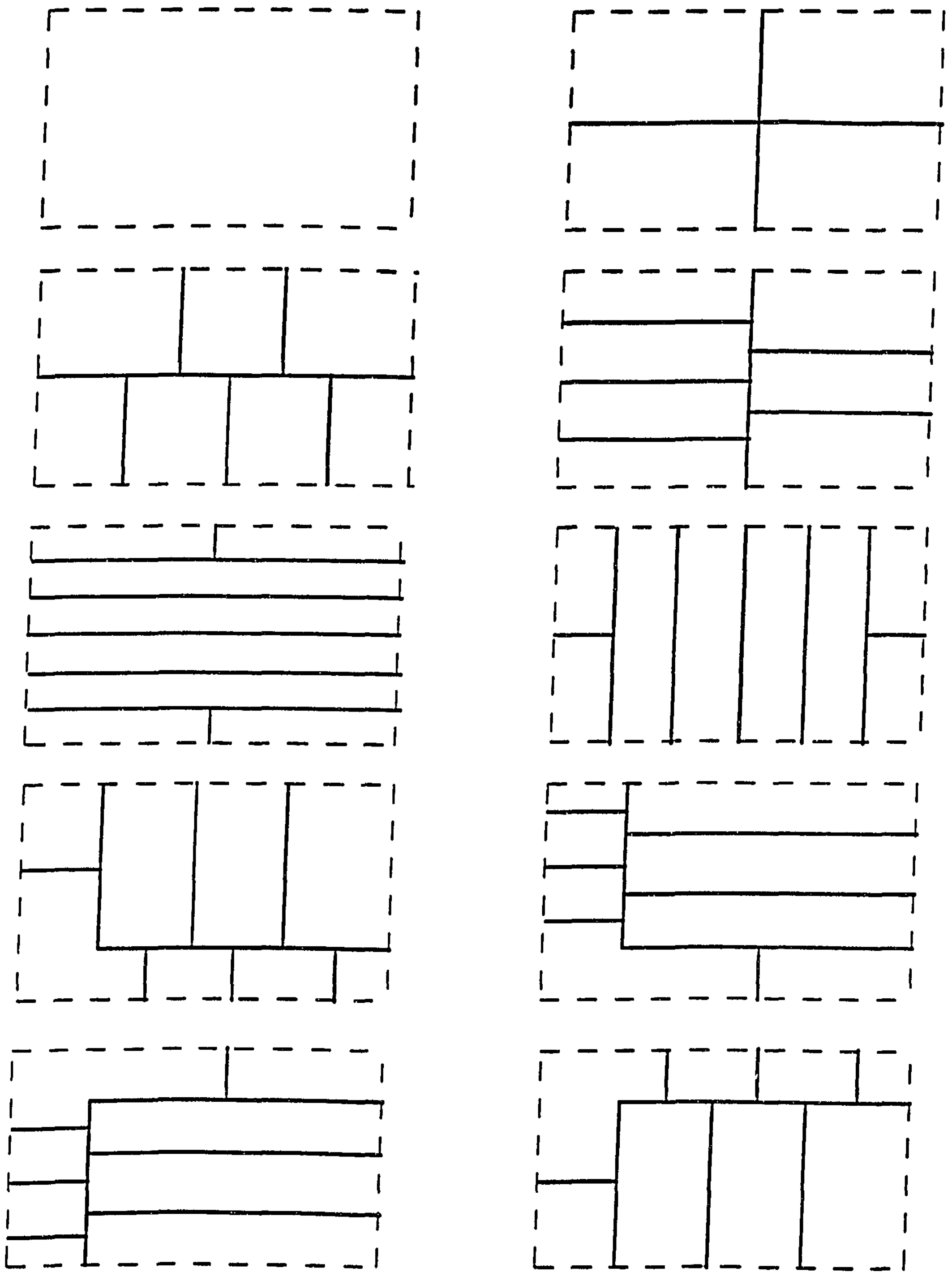


Figure 4.1 The Ten Topologies of S^*.

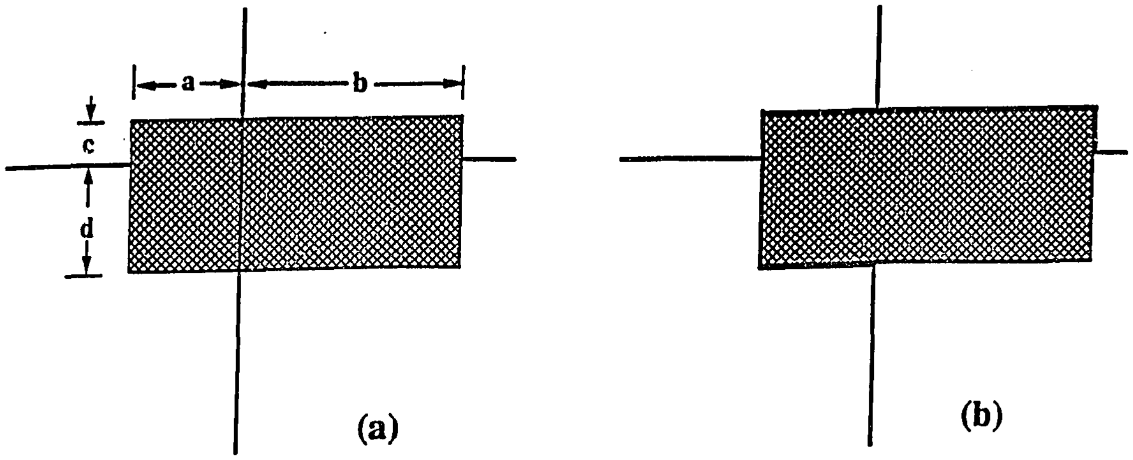


Figure 4.2 Case 1: Obstacle Intersects With a Cross Vertex.

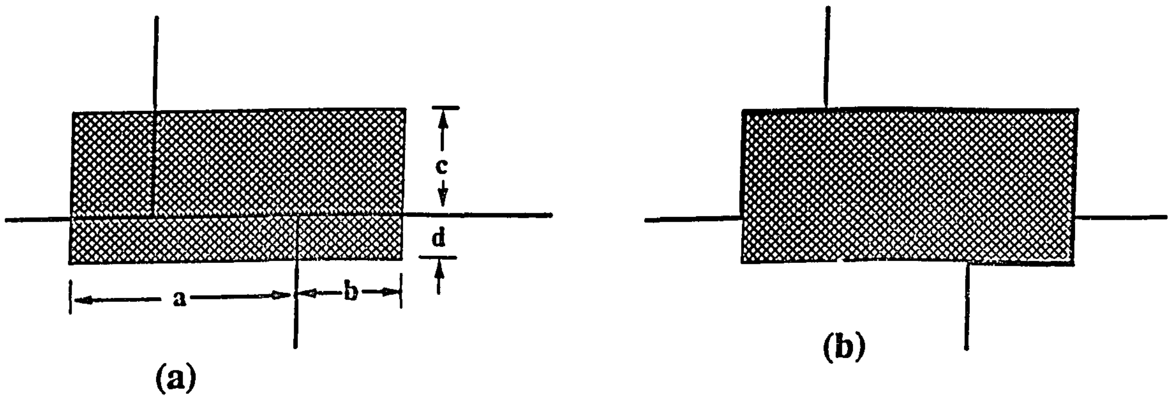


Figure 4.3 Case 2: Obstacle Intersects With Exactly Two T Vertices.

length is $c + a$, and

$$\begin{aligned} c \le d &\Rightarrow c \le \frac{1}{2}L(O_1), \\ a \le b &\Rightarrow a \le \frac{1}{2}W(O_1), \end{aligned}$$

therefore,

$$\begin{aligned} c + a &\le \frac{1}{2}L(O_1) + \frac{1}{2}W(O_1) \\ &\le \frac{1}{2}W(O_1) + \frac{1}{2}W(O_1) \\ &\le W(O_1). \end{aligned}$$

Note that, this case also covers the cases when the obstacle intersects with a horizontal edge, a vertical edge, or a corner vertex in a $SRST^*$.

Case 2 - *Obstacle intersects with exactly two T vertices:* In accordance with the topology of the SRST, whenever obstacle intersects exactly two vertices, they must be two consecutive T vertices of an interior edge. The legs of both the T vertices must be in the opposite direction. Figure 4.3(a) shows an obstacle intersecting with two T vertices. Without loss of generality, assume that $a \le b$ and $c \le d$. The modified tree S_1 is shown in Figure 4.3(b). It is clear that the additional cost is $c + a$. Derivation, similar to that of Case 1, shows that $c + a \le W(O_1)$.

Case 3 - *Obstacle intersects with multiple T vertices:* If the obstacle intersects with just the T vertices, and does not intersect with any corner vertex in S^* then all the T vertices must be on the same interior edge, as shown in Figure 4.4(a). The modified tree S_1 is shown in Figure 4.4(b). It is clear from the figure that in the worst case the additional cost will be a, which is less than $W(O_1)$.

Case 4 - *Obstacle intersects with a corner vertex:* When the obstacle intersects with only the corner vertex, then it can not intersect with an edge which is not the leg of a T vertex. If O_1 intersects with a corner vertex and one T vertex or no T vertex this case reduces to Case 1. We first consider that O_1 intersect with a corner vertex and two T vertices one on each of its legs, as shown in Figure 4.5. Next we consider that O_1 intersects with a corner vertex and multiple T vertices.

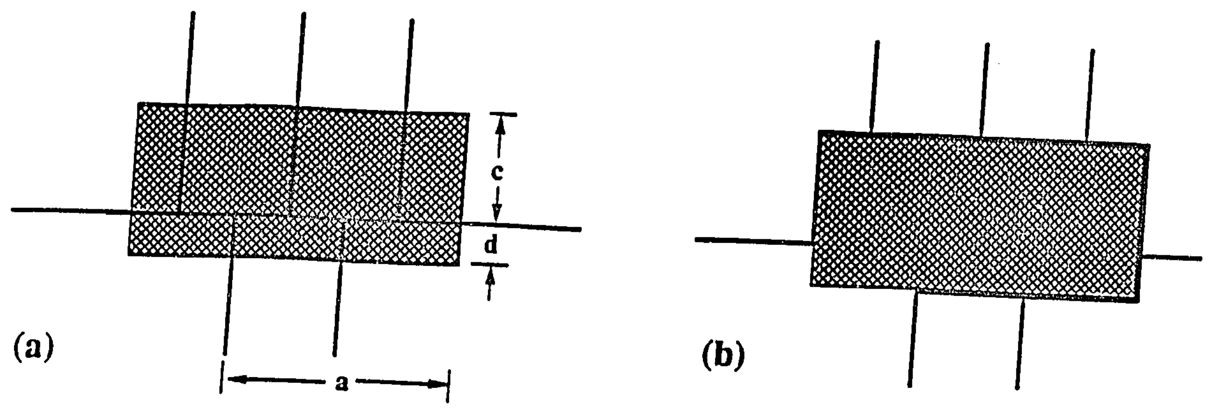


Figure 4.4 Case 3: Obstacle Intersects With Multiple T Vertices.

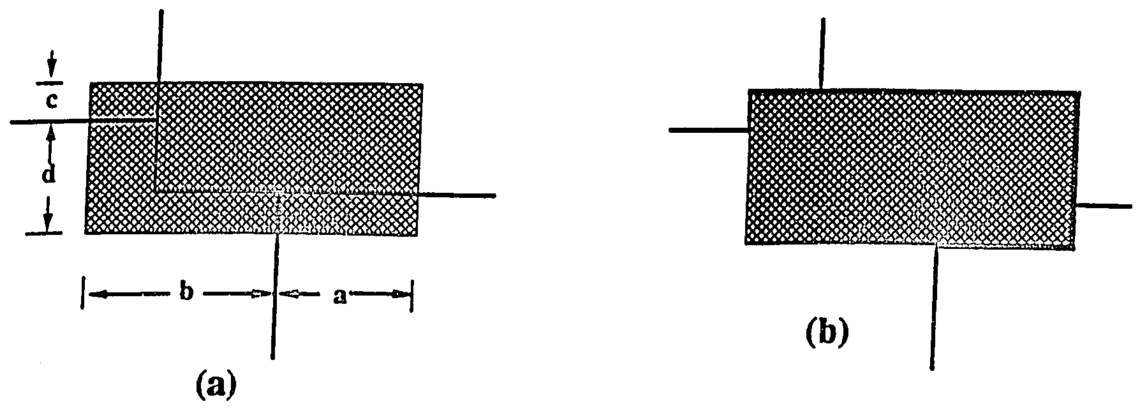


Figure 4.5 Case 4: Obstacle Intersects With a Corner Vertex.

For the SRST shown in Figure 4.5(a), let us assume that $a \leq b$ and $c \leq d$, the modified tree is shown in Figure 4.5(b). The additional cost is $c+a \leq W(O_1)$. Now, consider the case that O_1 intersects with a corner vertex and multiple T vertices as shown in Figure 4.5(a). The modified tree is shown in Figure 4.5(b). It is clear that in the worst case the additional cost is at most a and $a < W(O_1)$.
Case 5 - *Obstacle intersects with multiple parallel edges:* All the edges may either incident an interior line on one side on the boundary on the other side, or all the edges must be incident to the boundary on both the sides.

Case 5a - *Parallel edges anchor to boundary on one side and an interior edge on the other side:* Figure 4.6(a) shows a SRST in which parallel edges are intersected by an obstacle O_1. A modified tree is shown in Figure 4.6(b). It is clear that the additional cost in the worst case is less than $W(O_1)$.

Case 5b - *Parallel edges anchor to the boundary on each side:* The adjacent parallel edges must be connected either on top or bottom boundary. Let us consider the SRST shown in Figure 4.7(a), and modified tree shown in Figure 4.7(b). Note that, as the adjacent edges are connected through an edge either on top or bottom boundary, the additional cost is at most $W(O_1)$. ▨

We further extend the result and give a $O(n)$ time algorithm which finds an optimal Steiner tree S_1^*. The main idea behind *1-Obs-SRST* is to first find optimal or sub-optimal trees for each topology using the algorithm given in [9] or [1] considering that no obstacle is present. At next step obstacle O_1 is super-embedded with each tree. If O_1 intersects with any edge of a tree then that tree is modified to find the smallest tree (according to the topology of the tree), such that, it does not intersect with O_1 and the tree with the minimum cost is selected.

A *group* is a maximal sequence of demand points on the boundary of $\mathcal{R}$ connected by an exterior line. When obstacle O_1 intersects with an SRST S^*, it may disconnect many groups of demand points. Note that, it is possible to have more than one disconnected group only on two opposite sides of $\mathcal{R}$; the other two sides must have at most one disconnected group. When reconstructing the tree

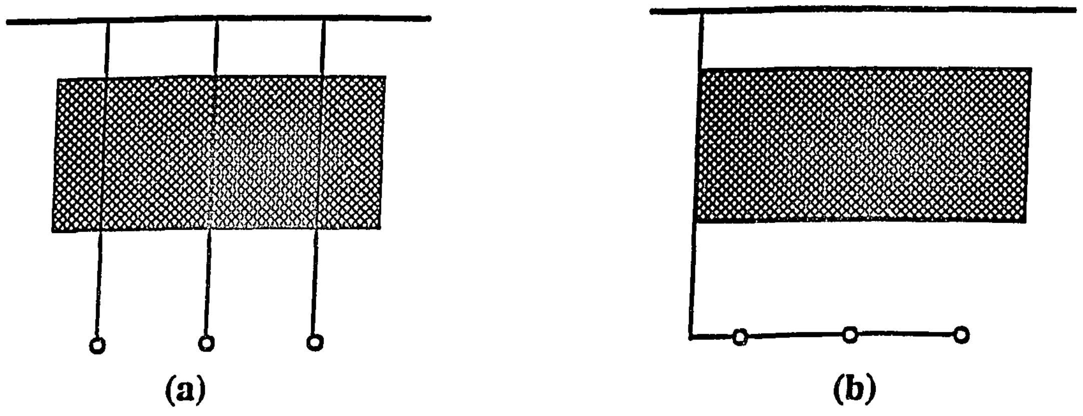


Figure 4.6 Case 5a: Parallel Edges Anchor to One Side and an Interior Edge on the Other Side.

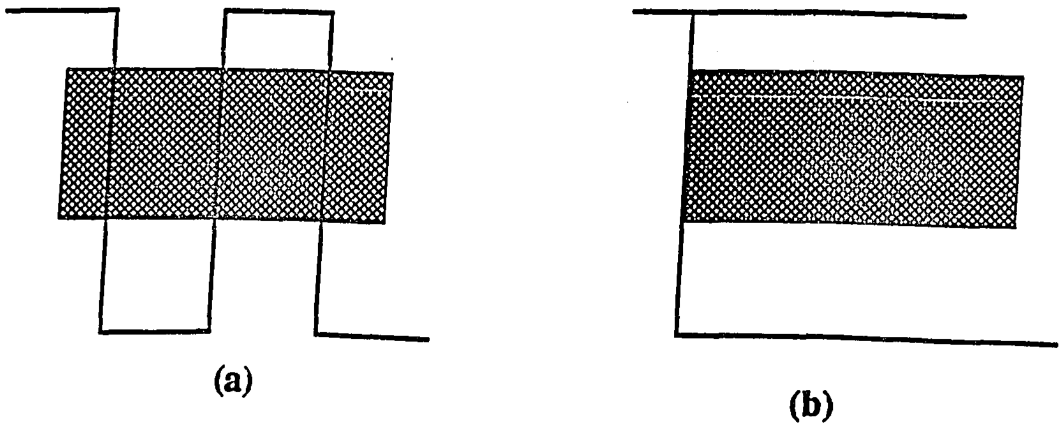


Figure 4.7 Case 5b: Parallel Edges Anchor to the Boundary on Each Side.

from the disconnected components, for any side of $\mathcal{R}$, all the adjacent groups may be connected together to form a bigger group which may be connected on either side to the rest of the groups, or we may form two groups, each of which can be connected to one side. Note that, even when there is just one disconnected group on any side of $\mathcal{R}$, it may still be split into two smaller groups. So, it is clear that we never have to consider more than two disconnected groups on each side of $\mathcal{R}$. Thus, we have a choice of making a big group or two smaller groups on each side of $\mathcal{R}$.

For a given set of points, algorithm *1-Obs-SRST* first finds the optimal or suboptimal RST S for each topology. If obstacle O_1 intersects with S then *1-Obs-SRST* makes modified trees as described earlier. It also constructs a tree as given for the previous theorem. The best tree for each topology is selected. The optimal tree is the one with the minimum cost among the best trees of all the topologies. It is clear that the above algorithm can be implemented in $O(n)$ time.

Theorem 10 *Algorithm* 1-Obs-SRST *finds a SRST S_1^* which does not intersect with obstacle O_1 in $O(n)$ time, where n is the number of demand points.*

So far, we have considered RSTO when only one obstacle is present inside $\mathcal{R}$. In the next section, we consider RSTO when k obstacles are present inside $\mathcal{R}$.

4.3 An Algorithm for Steiner Trees in Presence of n Obstacles

In this section, we give an algorithm, n-**RSTO**, which finds a *RST* S_n in the presence of n obstacles such that $\mu(S_n) \leq \mu(S^*) + \sum_{i=1}^{n} \max\{L(O_i), W(O_i)\}$. We show that n-**RSTO** can be implemented in polynomial time.

Algorithm n-**RSTO** takes a set of points V, every point lies on the boundary of a rectangle, and a set of obstacles $\mathcal{O}$, and returns a RST which spans over V without intersecting with any obstacle in $\mathcal{O}$. The basic intuition behind

n-**RSTO** is to first find an optimal Steiner tree, assuming that there are no obstacles present, using the algorithm given by Cohoon et al. [9]. Then introduce obstacles one by one and remove the edges intersecting with the obstacles. This results in decomposition of S^* into disjoint components. Next, we find paths to interconnect these disjoint components which forms a RST interconnecting all the points together.

At Step 1, algorithm finds the SRST assuming that there is no obstacle present. FindSRST is an implementation on the algorithm by Cohoon et al. [9] to find a SRST. $\mathcal{F}$ is a forest of trees, which is used to store the disconnected components of the SRST. To start up with, the tree is connected and thus $\mathcal{F}$ have just one element. At Step 2, it checks for the intersection of an obstacle with trees in $\mathcal{F}$. If O_i disconnects a tree in $\mathcal{F}$ then that tree splits into components and those components are stored in $\mathcal{F}$. Procedure **Intersects()** checks whether or not an edge e_{ij} intersects with obstacle O_k. Given a vertex v_j, **Neighbor()** finds the top, bottom, left and right neighbors of v_j. $\mathcal{SP}$ is a set of paths, at Step 3, we find all the paths from a point to all of its neighbors. Every time **SelectPaths()** selects set of $|\mathcal{F}| - 1$ paths from $\mathcal{SP}$. Procedure **Components()** finds the number of components in a given graph. At Step 4, n-SRST selects paths to connect the trees in $\mathcal{F}$ in RST with minimum cost and at Step 6 MinTree, a RST is returned. The detailed algorithm is given below.

Algorithm n-SRST(V, $\mathcal{O}$)

This algorithm takes a set V of vertices on the boundary of a rectangle and a set $\mathcal{O}$ of rectangular obstacle and finds a rectilinear short steiner tree.

Input: *Set of vertices,* $V = \{v_1, v_2, \ldots v_x\}$, *and*

set of obstacles, $\mathcal{O} = \{O_1, O_2, \ldots, O_n\}$.

Output: *A rectilinear Steiner tree* $MinTree = (V', E)$ *which spans over* V *without intersecting with any obstacle in* $\mathcal{O}$.

```
begin
  1. (⋆ find optimal steiner tree when no obstacle is present. ⋆)
     $\mathcal{F} \leftarrow$ FindSRST (V)
  2. (⋆ Introduce obstacles one by one and if some part of it gets disconnected
     then save the disconnected components. ⋆)
     for $k \leftarrow 1$ to $|\mathcal{O}|$ do
       for $m \leftarrow 1$ to $|\mathcal{F}|$ do
         for all $e_{ij}$ in $T_m$ do
           if Intersects($e_{ij}, O_k$) then
             let $T_x$ and $T_y$ be the components of $T_m - e_{ij}$
             $\mathcal{F} \leftarrow \mathcal{F} - T_m$
             $\mathcal{F} \leftarrow \mathcal{F} + T_x + T_y$
           end if
  3. (⋆ Find paths from all the interesting points to their neighbors in
         other Components. ⋆)
     $\mathcal{SP} \leftarrow \emptyset$
     for $k \leftarrow 1$ to $|\mathcal{F}|$ do
       for $j \leftarrow 1$ to $|V_k|$ do
         $\mathcal{N} \leftarrow$ Neighbor($v_j \in V_k$)
         if $v_j$ is corner vertex, T-vertex or an end vertex then
           for $m \leftarrow 1$ to $|\mathcal{N}|$ do
             $\mathcal{SP} \leftarrow \mathcal{SP} + P(v_j \in V_k, v_m \in \mathcal{N})$
  4. (⋆ Select a set of paths which connects all the components in $\mathcal{F}$
         with minimum cost. ⋆)
     $MinCost \leftarrow 0$
     for all possible $|\mathcal{F}| - 1$ path sets in $\mathcal{SP}$
       $\mathcal{X} \leftarrow$ SelectPaths ($|\mathcal{F}| - 1, \mathcal{SP}$)
       if Components ($\mathcal{F} \bigcup \mathcal{X}$) = 1 then
```

```
        $\mathcal{Z} \leftarrow \mathcal{F} - \mathcal{J} \bigcup \mathcal{X}$ (⋆ $\mathcal{J} \in \mathcal{F} \bigcup \mathcal{X}$ is not required for connectivity.⋆)
        if $\mu(\mathcal{Z}) < MinCost$ then
          $MinCost \leftarrow \mu(\mathcal{Z})$
          $MinTree \leftarrow \mathcal{Z}$
        end if
    5. (⋆ MinTree is a minimum tree with in the bound, return MinTree.⋆)
       return (MinTree)

end $n$-SRST
```

Theorem 11 *Algorithm n-***SRST** *finds a RST S_n in the presence of n obstacles such that $\mu(S_n) \leq \mu(S^*) + \sum_{i=1}^{n} \max\{L(O_i), W(O_i)\}$.*

A randomly generated instance of the problem and the solution produced by the algorithm is shown in Figure 4.8.

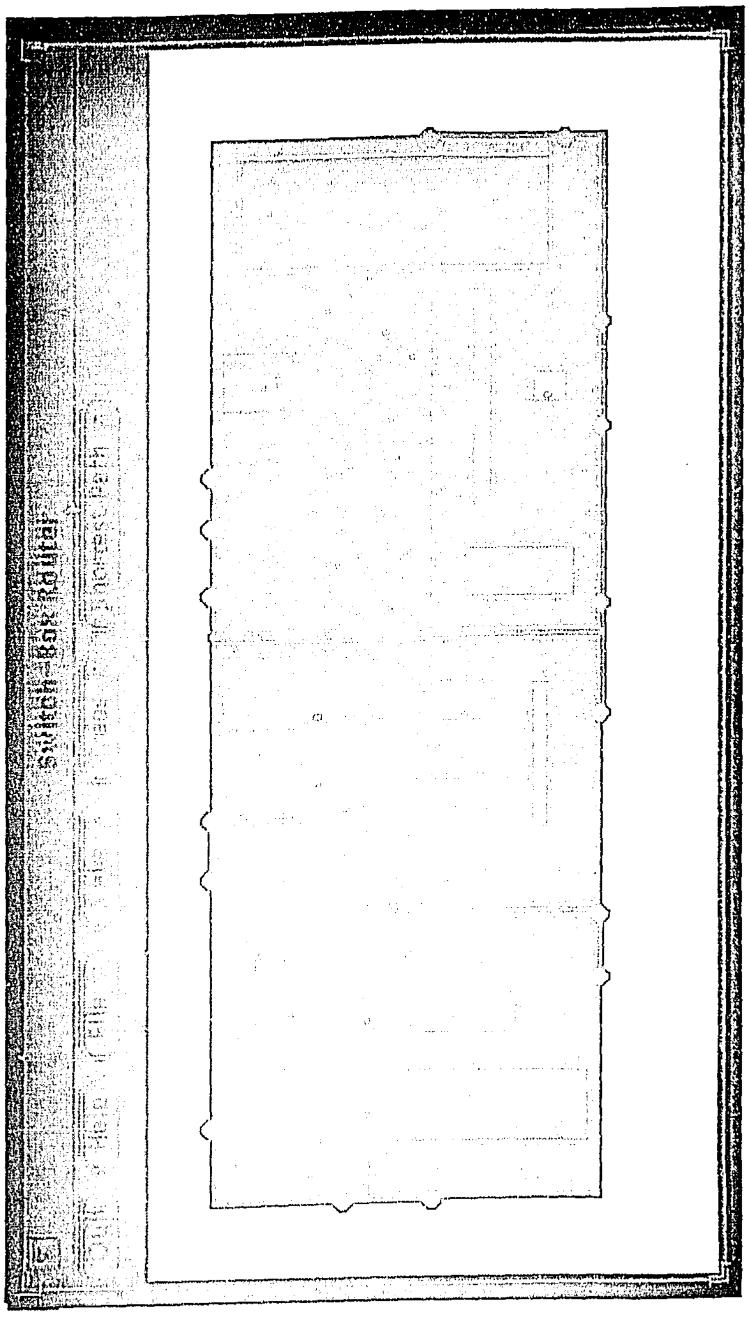


Figure 4.8 An Instance of RSTO and a Solution Produced by Algorithm n-SRST.

CHAPTER V

CONCLUSIONS AND DIRECTION OF FUTURE RESEARCH

In this thesis, we study two problems related to the planar routing of multi-terminal nets. The first problem is related to the VLSI Global routing and the second is related to the detailed area routing.

We consider the problem of routing k multi-terminal nets assigned to a routing layer, entirely on that layer and no sets of any nets intersect. A new approach is suggested which results in good solutions. Given k sets of points in the plane, we first find k non-intersecting spanning trees which each spans a given set, then these trees are rectilinearized one at a time. We show that the k-spanning tree problem (k-SPTP) is computationally very hard, even 2-SPTP is NP-complete. For 1-SPTP a solution always exists, but for $k \geq 2$, k-SPTP may not be solvable. We have investigated the necessary and sufficient conditions for the existence of a solution for a 2-SPTP. We establish the conditions under which the problem can be solved. We present an efficient algorithm which produces solutions very close to the optimal.

Currently, we are investigating the necessary and sufficient conditions for the existence of a solution to a k-SPTP, $k > 2$. Our preliminary studies show that general k-SPTP is very hard problem. The necessary and sufficient conditions for 3-SPTP are much more complicated than the 2-SPTP.

We study the problem of finding minimum Steiner tree in presence of obstacles when the terminals lie on the boundary of the rectangle (RSTO) and present two results. Our first contribution is an exact solution for finding the shortest rectilinear Steiner tree in presence of an obstacle when the terminals lie on the boundary of the rectangle. Secondly, we give an approximation algorithm for RSTO in presence of n obstacles. The bound of our algorithm is less than or equal to $\mu(S^*) + \sum_{i=1}^{n} \max\{L(O_i), W(O_i)\}$, where S^* is a Shortest Rectilinear Steiner Tree when no obstacle in present and $L(O_i)$ $(W(O_i))$ represent the length (width) of obstacle O_i.

Consider two routing layers are available. Let $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$ be a set of obstacles, where each O_i is properly contained in $\mathcal{R}$ (switchbox), and let layer 1 and layer 2 be represented by L_1 ans L_2 respectively. An obstacle O_i may exist on L_1, L_2, or both. The following problems remain open:

Problem 1 *Given a set of terminals on the boundary of $\mathcal{R}$ and a set of obstacles $\mathcal{O}$, where each $O_i \in \mathcal{O}$ lies either on L_1 or L_2, find a SRST S_k^*. What is the upper bound for S_k^* ? (Non-intersecting obstacles.)*

Problem 2 *Given a set of terminals on the boundary of $\mathcal{R}$ and a set of obstacles $\mathcal{O}$, where each $O_i \in \mathcal{O}$ lies either on L_1, L_2, or both, find a SRST S_k^*. What is the upper bound for S_k^* ? (Obstacles may intersect.)*

Figure 5.1 and Figure 5.2 show instances of Problems 1 and 2, respectively.

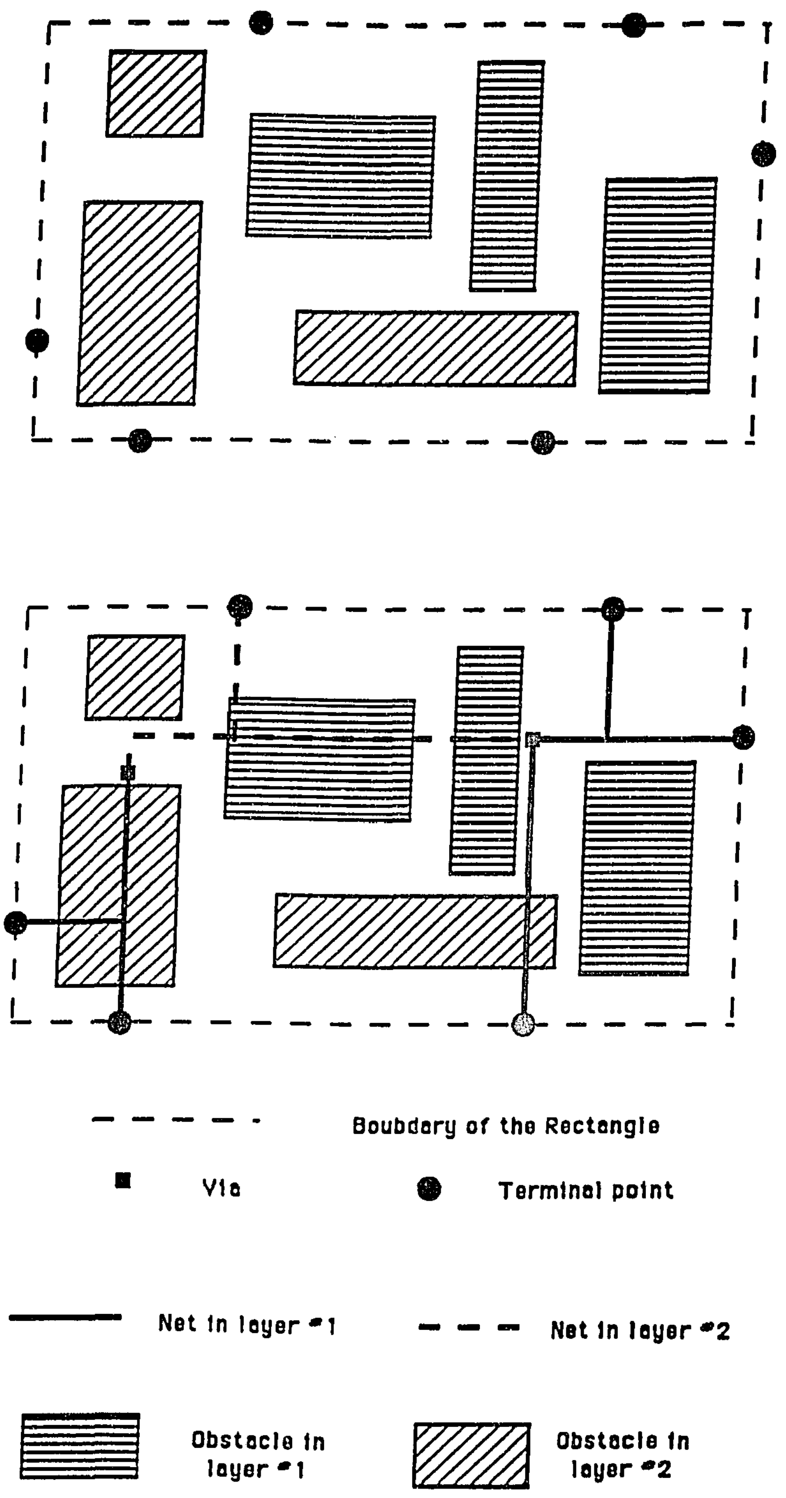


Figure 5.1 Two Layer SRTO With Non-Intersecting Obstacles.

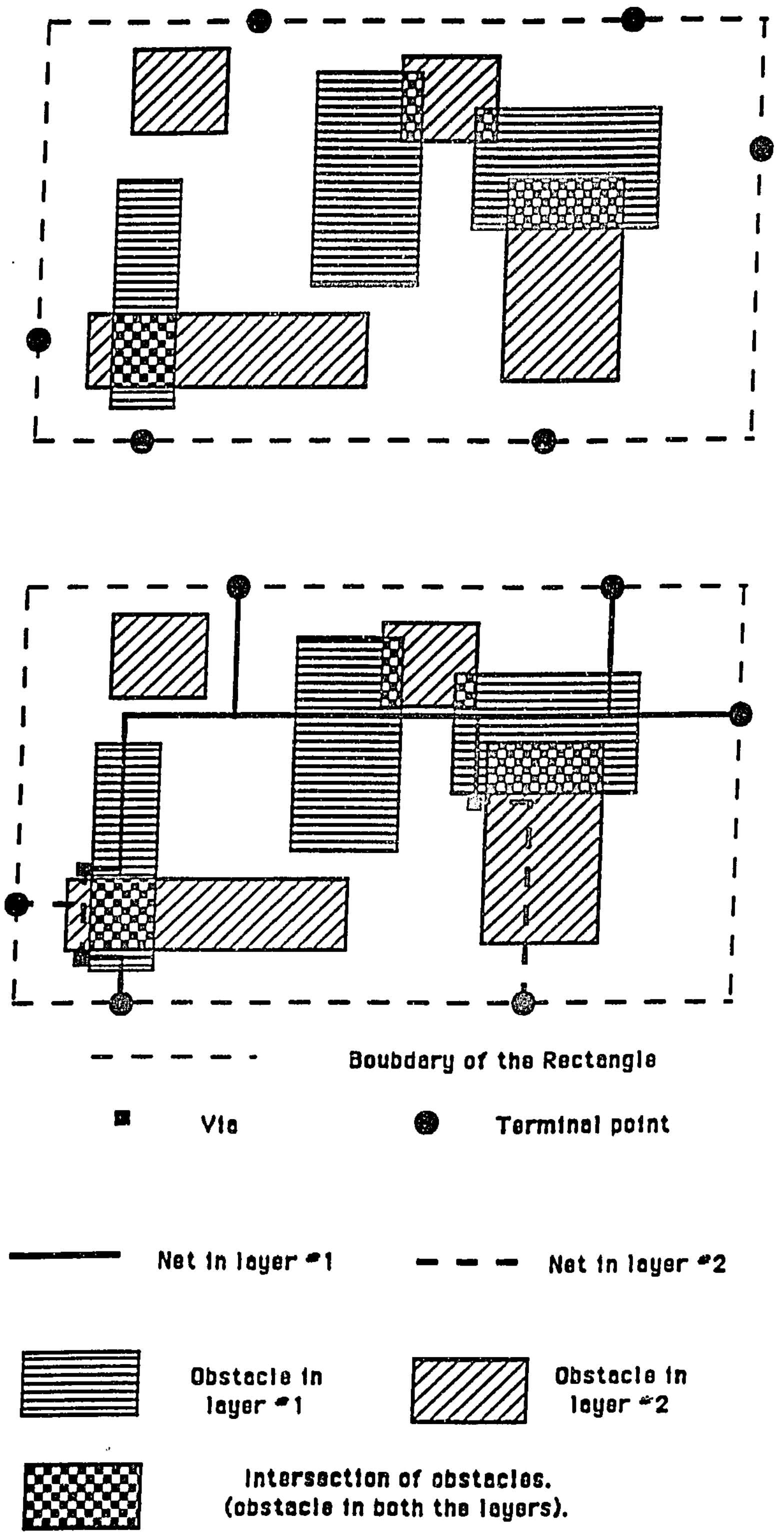


Figure 5.2 Two Layer RSTO With Intersecting Obstacles.

REFERENCES

[1] P. K. Agarwal and M-T. Shing, "Algorithms for special cases of rectilinear Steiner trees: I. Points on the boundary of a rectilinear rectangle," *Networks,* Vol. 9, No. 20, 1990, pp. 453-485.

[2] A.V. Aho, M. R. Gary, and F. K. Hwang, "Rectilinear Steiner trees: Efficient special case algorithms," *Networks,* Vol. 7, 1977, pp. 37-58.

[3] F. Aurenhammer, "Vornoi diagrams – a survey of a fundamental geometric data structure," *ACM Computing Surveys,* Vol. 23, No. 3, September 1991, pp. 345-405.

[4] B.S. Baker and R. Y. Pinter, "An algorithm for the optimal placement and routing of a circuit within a ring of pads," *24th Annual Symposium on Foundation of Computer Science,* Nov. 1983, pp. 360-370.

[5] M. Bern, "Faster exact algorithms for Steiner trees in planner networks," *Networks,* Vol. 20, 1990, pp. 109-120.

[6] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs and graph planarity using PQ-trees algorithm," *Journal of Computer and System Sciences,* Vol. 13, 1979, pp. 335-379.

[7] R. Carden and C-K Cheng, "Feasibility estimation and cost optimization for multichip module technologies," in the proceedings of *4th Annual IEEE International ASIC Conference,* 1991, P9-2.

[8] D. Cheriton and R. Tarjan, "Finding minimum spanning trees," *SIAM Journal of Computing,* Vol. 5, No. 4, 1977, pp. 724-742.

[9] J. P. Cohoon, D. S. Richards, and J. S. Salowe, "An optimal Steiner tree algorithm for a net whose terminals lie on the perimeter of a rectangle," *IEEE Transactions on Computer-Aided Design,* Vol. 9, No. 4, April 1990, pp. 398-407.

[10] R. Condamoor and I. Tollis, "A new heuristic for rectilinear Steiner trees," in the *Proceedings of International Symposium on Circuits and Systems,* 1990, pp. 1676-1679.

[11] P. J. de Rezende, D. T. Lee, and Y. F. Wu, "Rectilinear shortest paths in the presence of rectangular barriers," in the *Proceedings of ACM Symposium Computation Geometry,* 1987, pp.204-213.

[12] D. Dolev, K. Karplus, A. Siegel, A. Strong, and J. D. Ullman, "Optimal wiring between rectangles," in the *Proceedings of the 13th Annual ACM Symposium on Theory of Computing,* May 1981, pp. 312-317.

[13] C. P. Gabor, W.-L. Hsu, and K. J. Supowit, "Recognizing circle graphs in polynomial time," in the *Proceedings of 26th IEEE Symposium on Foundation of Computer Science,* 1989, pp. 106-116.

[14] H. Gabow, Z. Galil, T. Spencer, and E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected graphs," *Combinatorica,* Vol. 6, No. 2, 1986, pp. 109-122.

[15] M. R. Garey and D. S. Johnson, "The rectilinear Steiner tree problem is NP-Complete," *SIAM Journal of Applied Mathematics,* Vol. 32, No. 4, Jan. 1977, pp. 37-58.

[16] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, "The complexity of coloring circular arcs and chords," *SIAM Journal of Algorithms and Discrete Mathematics,* Vol. 1, 1980, pp. 216-228.

[17] F. Gavril, "Algorithms for a maximum clique and a maximum independent set of a circle graph," *Networks,* Vol. 3, 1973, pp. 261-173.

[18] E. N. Gilbert and H. O. Pollak, "Steiner Minimal Trees," *SIAM Journal of Applied Mathematics,* Vol. 16, No. 1, 1968, pp. 1-20.

[19] P. C. Gilmore and A. J. Hoffman, "A characterization of comparability graphs and of interval graphs," *Canadian Journal of Mathematics,* Vol. 16, 1964, pp. 539-548.

[20] M. C. Golumbic, "Algorithmic graph theory and perfect graphs," Academic Press, New York, 1980.

[21] N. Hasan, G. Vijayan, and C. Wong, "A neighborhood Improvement algorithm for rectilinear Steiner trees," in the *Proceedings of International Symposium on Circuits and Systems,* 1990, pp. 2869-2872.

[22] J-M. Ho, G. Vijayan, and C. K. Wong, "A new approach to the rectilinear Steiner tree problem," *26th ACM/IEEE Design Automation Conference,* June 1989, pp. 161-166.

[23] J-M. Ho, G. Vijayan, and C. K. Wong, "Constructing the optimal rectilinear Steiner tree derivable from a minimum spanning tree," *Proceedings of IEEE International Conference on Computer-Aided Design,* Nov. 1989, pp. 5-8.

[24] J. Ho, D. Lee, C. Chang, and C. Wang, "Minimum diameter spanning trees and related problems," *SIAM Journal of Computing,* Vol. 20, No. 5, October 1991, pp. 987-997.

[25] W. L. Hsu, "Maximum weight clique algorithms for circular-arc graphs and circle graphs," *SIAM Journal of Computing,* Vol. 14, No. 1, Feb. 1985, pp. 160-175.

[26] T. Hu and M. Shing, "A decomposition algorithm for circuit routing," *VLSI Circuit Layout: Theory and Design,* IEEE Press, 1985, pp. 144-152.

[27] F. K. Hwang, "On Steiner minimal trees with rectilinear distance," *SIAM Journal of Applied Mathematics,* Vol. 30, No. 1, Jan. 1976, pp. 104-114.

[28] J. Jájá and S. Alice Wu, "On routing two-terminal nets in the presence of obstacles," *IEEE Transactions on Computer-Aided Design,* Vol. 8, No. 5, May 1989, pp. 563-570.

[29] M. R. Kramer and J. van Leeuwen, "Wire-routing is NP-complete," Technical Report RUU-CS-82-4, Department of Computer Science, University of Utrecht, Netherlands, Feb. 1982.

[30] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," *Proceedings of the American Mathematical Society,* Vol. 7, 1956, pp. 48-50.

[31] C. Leiserson and F. Maley, "Algorithms for routing and testing routability of planar VLSI layouts," in the *Proceedings of the 17th Annual Symposium on Theory of Computing,* 1985, pp. 69-78.

[32] C. E. Leiserson and R. Y. Pinter, "Optimal placement for river routing," *SIAM Journal of Computing*, Vol. 12, No. 3, pp. 447-462.

[33] G. Leonidas, J. Hershberger, and J. Soneyunk, "Compact interval trees: A data structure for convex-hulls," *International Journal of Computational Geometry & Applications,* Vol. 1, No. 1, 1991, pp. 1-22.

[34] K-F Liao, M. Sarrafzadeh, and C. Wong, "Single-layer global routing," in the *Proceedings of the 4th Annual IEEE International ASIC Conference,* 1991, P14-4.

[35] F. Maley, "Single-layer wire routing and compaction," The MIT Press, Cambridge, Massachusetts, 1990.

[36] M. Marek-Sadowska, "Route planner for custom chip design," in the *Proceedings of the IEEE International Conference on Computer-Aided Design,* 1986, pp. 246-249.

[37] A. Pneuli, S. Even, and A. Lempel, "Transitive orientation of graphs and identification of permutation graphs," *Canadian Journal of Mathematics,* Vol. 23, 1971, pp. 160-175.

[38] R. C. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal,* Vol. 36, 1957, pp. 1389-1401.

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

[40] R. Rivest, A. Baratz, and G. Miller, "Provably good channel routing algorithms," *Carnegie-Mellon Conference on VLSI Systems and Computations,* 1981, pp. 153-159.

[41] A. Siegel and D. Dolev, "The separation for general single-layer wiring barriers," *Carnegie-Mellon Conference on VLSI Systems and Computations,* 1981, pp. 143-152.

[42] J. Spinard, "On comparability and permutation graphs," *SIAM Journal of Computing,* Vol. 14, No. 3, Aug. 1985, pp. 658-670.

[43] T. Szymanski, "Dogleg channel routing is NP-complete," *IEEE Transactions on Computer-Aided Design of Circuits and Systems,* Vol. CAD-4, No. 1, 1985, pp. 31-41.

[44] R. Tarjan, "Data structures and network algorithms," *Society for Industrial and Applied Mathematics,* 1983, pp. 123-145.

[45] M. Tompa, "An optimal solution to a wire-routing problem," *Journal of Computer and Systems Sciences,* Vol. 23, No. 2, 1981, pp. 127-150.

[46] D. Tritsch, "Interconnecting networks in the plane: The Steiner case," *Networks,* Vol. 20, 1990, pp. 93-108.

[47] A. Yao, "An $O|E|\log\log|V|)$ algorithm for finding minimum spanning trees," *Information Processing Letters,* Vol. 4, 1975, pp. 21-23.