Graph Theory Based Routing Algorithms

Bo Wu
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses
Part of the Computer Sciences Commons, and the Mathematics Commons

Recommended Citation
Wu, Bo, "Graph Theory Based Routing Algorithms" (1992). Master's Theses. 957.
http://scholarworks.wmich.edu/masters_theses/957
GRAPH THEORY BASED ROUTING ALGORITHMS

by

Bo Wu

A Thesis
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
In this thesis, we study two VLSI layout problems.

First, we investigate the minimum-bend single row routing problem in which the objective function is to minimize the number of doglegs (or bends) per net. Our approach uses a graph theoretic representation in which an instance of the single row routing problem is represented by three graphs: an overlap graph, a containment graph, and an interval graph. Using this graph representation, we develop three algorithms for the minimum-bend single row routing problem. We show that our algorithms have very tight performance bounds.

Second, we present a three layer over-the-cell router (ICR-3) for the standard cell design style based on a new cell model which assumes that terminals are located in the center of the cells in layer M2. This model is similar to the one currently being developed for three layer cell libraries in industry. We have implemented ICR-3 in C on a SUN SPARCstation 1+ and tested it on several benchmarks including PRIMARY I and PRIMARY II from Microelectronics Center of North Carolina (MCNC). Our router out performs all existing routers. We show that ICR-3 produces results which are better (on the average) by 58% as compared to a two layer over-the-cell (20TC) router and 47% as compared to a conventional three layer channel (3CRP) router.
ACKNOWLEDGMENTS

How does one acknowledge the many people who contribute to the fulfillment of a dream? The following are just a few who contributed in the completion of this thesis.

First, I thank my advisor, Dr. Naveed A. Sherwani. I could not have made so much progress without his guidance and help in my school years. Second, to him and other members of my committee, Dr. Fred Boals and Dr. Ajay Gupta, I extend my sincere appreciation for their supervision and support. I would like to thank all the faculty in the Department of Computer Science and Western Michigan University, Kalamazoo who have ever provided opportunity, encouragement, and support to me.

I also want to mention my fellow students in our night group, Arun, Bhingarde, Burman, Chander, Jehangir, Moazeem, Sivakumar, Wasim, and others. I like to work with them and thank them for their encouragement and help.

Finally, to my only family member, my wife Ling Lu, I owe a debt of gratitude. I thank her for her love and continuing to help me to do my research. I know it is the happiest thing for her to see the completion of my thesis and my graduation.

Bo Wu
To my wife,  
Ling, for her love and help.
Graph theory based routing algorithms

Wu, Bo, M.S.
Western Michigan University, 1992
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>v</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vi</td>
</tr>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Routing Problems in VLSI</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 Single Layer Routing Problems</td>
<td>3</td>
</tr>
<tr>
<td>1.1.2 Two Layer Routing Problems</td>
<td>4</td>
</tr>
<tr>
<td>1.1.3 Three Layer Routing Problems</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Single Row Routing Problem</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Over-The-Cell Channel Routing Problem</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Objectives of Thesis Research</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Thesis Organization</td>
<td>7</td>
</tr>
<tr>
<td>II. A GRAPH THEORY MODEL FOR ROUTING PROBLEMS</td>
<td>8</td>
</tr>
<tr>
<td>2.1 Preliminaries</td>
<td>8</td>
</tr>
<tr>
<td>2.2 A Graph Model</td>
<td>11</td>
</tr>
<tr>
<td>III. MINIMUM-BEND SINGLE ROW ROUTING</td>
<td>14</td>
</tr>
<tr>
<td>3.1 Routing Based on Graph Decomposition</td>
<td>17</td>
</tr>
<tr>
<td>3.1.1 Routing Based on Independent Set Decomposition</td>
<td>17</td>
</tr>
<tr>
<td>3.1.2 Routing Based on Maximum Bipartite Subset Decomposition</td>
<td>25</td>
</tr>
<tr>
<td>3.1.3 Routing Based on Maximum Non-Containment Subset Decomposition</td>
<td>28</td>
</tr>
</tbody>
</table>
Table of Contents – Continued

3.2 Extension of the Proposed Algorithms ........................................... 35
3.3 Experimental Results ................................................................. 39
3.4 Summary .................................................................................... 41

IV. OVER-THE-CELL ROUTING BASED ON NEW CELL MODEL .... 47

4.1 The Existing and the New Cell Models ...................................... 49
  4.1.1 Existing Model ................................................................ 50
  4.1.2 New Cell Model ............................................................... 50
4.2 An Overview of ICR-3 ............................................................... 51
4.3 Detailed Description of the Algorithm ........................................ 55
  4.3.1 Net Classification ............................................................ 55
  4.3.2 Net Weights and Selection for M2 Layer ...................... 56
  4.3.3 Decomposition of Multi-Terminal Nets ......................... 59
  4.3.4 Terminal Assignments of Critical Nets ....................... 59
  4.3.5 Terminal Assignments of Non-Critical Nets ................. 66
  4.3.6 River Routing of M2 OTC Area .................................... 68
  4.3.7 Net Selection for M3 Layer ............................................. 68
  4.3.8 The Performance Bounds of ICR-3 ............................... 73
4.4 Formal Statement of the Algorithm ........................................... 74
4.5 Experimental Results ................................................................. 77
4.6 Summary .................................................................................... 80

V. CONCLUSIONS AND FUTURE RESEARCH .............................. 82
  5.1 Conclusions ............................................................................ 82
  5.2 Future Research ...................................................................... 82

REFERENCES .................................................................................. 83
LIST OF TABLES

3.1 Experimental Results: The Average Maximum Number of Doglegs Per Net .................................................. 42

3.2 Experimental Results: Running Time .............................................. 42

4.1 Experimental Results for PRIMARY 1: Channel Height ................ 78

4.2 Experimental Results for PRIMARY 1: Via Minimization .............. 80

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LIST OF FIGURES

1.1 A Single Row Routing Instance ............................................................ 6
2.1 Graph Representation for a Net List .................................................... 13
3.1 The Net Length and the Doglegs .......................................................... 16
3.2 The Insertion of Nets ................................................................. 20
3.3 The Insertion of an Independent Set ................................................... 24
3.4 Illustration of Algorithm $\text{find-MNS}$ .............................................. 30
3.5 An Example of a Valid Insertion ........................................................ 38
3.6 A Layout Instance Created by the Four Algorithms ............................. 40
3.7 Dogleg Distribution of Nets ............................................................. 43
3.8 Relationship Between Maximum Number of Doglegs Per Net and Clique Size ............................................................. 44
3.9 Relationship Between Maximum Number of Doglegs Per Net and Problem Size ............................................................. 45
4.1 Cell Models ................................................................. 52
4.2 A Routing Example Created by ICR-3 ................................................ 53
4.3 An Example of the Permutation Graph .............................................. 58
4.4 Terminal Assignments of Critical Nets ................................................ 69
4.5 Example of Overlap Graph ............................................................... 70
4.6 Complete Routing Created by ICR-3 ................................................... 79
CHAPTER I

INTRODUCTION

In the combinatorial sense, the layout problem is a constrained optimization problem. We are given a description of a circuit—most often as a netlist, which is a description of switching elements and their connecting wires. We look for an assignment of geometric coordinates of the circuit components—in the plane or in one of a few planar layers—that satisfies the requirements of the fabrication technology (sufficient spacing between wires, restricted number of wiring layers, and so on) and that minimizes certain cost criteria (the area of the smallest circumscribing rectangle, the length of the longest wire, and so on).

Practically all versions of the layout problem as a whole are intractable; that is, they are NP-hard. Thus, we have to resort to heuristic methods. One of these methods is to break up the problem into subproblems, which are then solved one after the other. Almost always, these subproblems are NP-hard as well, but they are more amenable to heuristic solution than is the layout problem itself. Each one of the layout subproblems is decomposed in an analogous fashion. In this way, we proceed to break up the optimization problems until we reach primitive subproblems. These subproblems are not decomposed further, but rather are solved directly, either optimally—if an efficient optimization algorithm exists—or approximately.

The most common way of breaking up the layout problem into subproblems is first to do component placement, and then to determine the approximate course of the wires in a global-routing phase. This phase may be followed by a topological compaction that reduces the area requirement of the layout, after which a detailed-routing phase determines the exact course of the wires without changing the layout area. After detailed routing, a geometric-compaction phase may further reduce the
area requirement of the layout. This whole procedure may be done hierarchically, starting with large blocks as circuit components, which are themselves laid out recursively in the same manner. This recursive process may be controlled by algorithms and heuristics that allow for choosing among layout alternatives for the blocks such that the layout area of the circuit is minimized. Exactly how a given version of the layout problem is broken up into subproblems depends on both the design and the fabrication technology. For instance, in standard-cell design, the detailed-routing phase essentially reduces to channel routing.

1.1 Routing Problems in VLSI

Routing is an important problem in circuit layout and between 20% to 30% 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 a rectilinear shape. The routing task is usually divided into two steps: (1) global routing, and (2) channel routing.

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

In VLSI design, channel 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 also has been used by industry. The routing may be done in such a manner that each net consists of vertical and horizontal wire segments. Then, one layer is used for horizontal wire segments while the other layers are 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 channel routing problem depends on the number of layers being used for routing. Thus based on number of layers, we have (a) one layer, (b) two layer and (c) three layer routing problems.

### 1.1.1 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 two main problems in single layer routing are: (1) river routing, and (2) single row 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.

On the other hand, single row routing is a single layer routing problem in which terminals lie in a single line. Single row routing not only is very important in VLSI chip layout, but also play an important role in printed circuit boards (PCB) layout.
1.1.2 Two Layer Routing Problems

The two layer channel routing problem can be stated as follows. Given 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 $W$.

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 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.

1.1.3 Three Layer Routing Problems

Three metal layer technology has changed the traditional channel routing problem dramatically. In three layer environment, we have two metal layers for horizontal wires and one metal layer for vertical wires. This allows us to reduce the channel height almost by half comparing with the two layer routing environment.

In addition, three metal layer technology also gives us two empty routing regions in over-the-cell areas. A new channel routing technique – over-the-cell channel routing – is trying to utilize the over-the-cell areas. And the research work of this thesis shows that by using over-the-cell routing technique, we can achieve channelless layout.

1.2 Single Row Routing Problem

The single-row routing problem can be defined as follows. We are given a set of two-terminal or multi-terminal nets defined on a set of evenly spaced terminals on a real line called the node axis. Without loss of generality, it can
be assumed that the node axis is a horizontal line. The interconnection for the nets is realized by means of non-crossing paths. Each path consists of horizontal and vertical line segments on a single layer, such that no two paths cross each other. Moreover, no path is allowed to intersect a vertical line more than once, i.e., backward moves of nets are not allowed. Figure 1.1 explains the notation and shows an example of a possible solution for a SRRP.

The area above the node axis is called the upper street while the area below the node axis is called the lower street. The number of horizontal tracks available for routing in the upper street is called upper street capacity. Similarly the number of horizontal tracks available in the lower street is called the lower street capacity. Due to symmetry in single row routing, upper street capacity is usually equal to lower street capacity. For a given realization, the number of the horizontal tracks needed in the upper street is called the upper street congestion \( (C_{us}) \) and the number of horizontal tracks needed in the lower street is called the lower street congestion \( (C_{ls}) \). The term dogleg is used to describe a bend in a net, when it makes an interstreet crossing. The between-nodes congestion \( C_B \) of a realization is the maximum number of interstreet crossings between a pair of adjacent terminals.

The objective function considered most often is to minimize the maximum of upper and lower street congestions. To minimize the separation between the two adjacent terminals is also necessary.

1.3 Over-The-Cell Channel Routing Problem

Over-the-cell channel routing is a superproblem of the channel routing problem. Traditionally, algorithms for over-the-cell channel routing consist of four steps:

1. Net selection: choosing suitable nets to route over the cells;
2. Over-the-cell routing: finding an over-the-cell layout for selected nets;
Figure 1.1 A Single Row Routing Instance.
3. Channel segment assignment: choosing net segments in the channel; and

The objective of over-the-cell channel routing is to minimize the channel height.

1.4 Objectives of Thesis Research

The main objectives of this thesis research are to develop new efficient algorithms for the single row routing and over-the-cell routing problems. The emphasis is on development of new graph theoretic algorithms for routing problems. This facilitates the use of graph theoretic techniques for developing various algorithms for single row routing and over-the-cell channel routing problems. Based on the graph theoretic approach, we develop several decomposition schemes for single row routing and present several performance bounded minimum-bend single row routing algorithms. The graph theoretic approach also allows us to obtain some optimal algorithms for over-the-cell routing. Finally, we investigate the future research directions in single row routing and over-the-cell routing.

1.5 Thesis Organization

This thesis is organized into five chapters. In Chapter II, we explain a graph theory model for routing problems. Several minimum-bend single row routing algorithms are presented in Chapter III. In Chapter IV, we introduce a new over-the-cell router based on the new cell model. Open problems and future research directions in single row routing and over-the-cell routing are proposed in Chapter V.
CHAPTER II

A GRAPH THEORY MODEL FOR ROUTING PROBLEMS

In this chapter we review the graph formulation of the routing problems. We begin with a review of relevant definitions from graph theory.

2.1 Preliminaries

A graph is a pair of sets $G = (V, E)$, where $V$ is a set 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 $\text{Adj}(v)$. 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 $\phi$ 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 $e_i$ 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 a subgraph \( C \) such that \( E(C) = V(C) \times V(C) \).

A directed graph is a pair of set \((V, \bar{E})\), where \( V \) is a set of vertices and \( \bar{E} \) is a set of ordered pairs of distinct vertices, called directed edges or arcs. We use the notation \( \bar{G} \) for a directed graph. An arc \( \bar{e} = \{u, v\} \) is incident with \( u \) and \( v \), the vertices \( u \) and \( v \) are the head and tail of \( \bar{e} \), respectively; \( \bar{e} \) 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 \( \bar{G} = (V, \bar{E}) \) 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 \( \bar{G} \) 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 \( \bar{G} \). 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 \( \text{root}(T) \). 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.

There are several classes of graphs that we will use in the development of properties of routing problems; here we briefly review definitions related to these classes.
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 are 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 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 [14]. Linear time algorithms are known for recognition, maximum clique, maximum independent set problems among others for this class of graphs.

A graph $G$ is called a permutation graph if 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 [14, 37].

Circle graphs, defined as the intersection graph of chords of a circle, can be recognized in polynomial time and polynomial algorithms are also known for maximum clique and maximum independent set problems on circle graphs [22].

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.

2.2 A Graph Model

Let $R$ be a set of evenly spaced terminals. A net $N$ is defined to be a subset of nodes in $R$, i.e., $N \subseteq R, |N| \geq 2$. $N$ is called a simple net if $|N| = 2$, otherwise it is called a multi-net. Let $L = \{N_1, N_2, \ldots, N_n\}$ be a set of nets defined on $R$. Each net $N_i$ can be uniquely specified by two distinct terminals $l_i$ and $r_i$, called the left touch point and the right touch point, respectively, of $N_i$. Abstractly, a net can be considered as an interval bounded by left and right touch points. Thus for a given set of nets, an interval diagram depicting each net as an interval can be easily constructed. Given an interval diagram corresponding to a set of nets, two graphs representing the routing problem can be defined as follows.

Define an overlap graph $\bar{G}_O = (V, \bar{E}_O)$,

$$V = \{v_i \mid v_i \text{ represents interval } I_i \text{ corresponding to net } N_i\}$$

$$\bar{E}_O = \{(v_i, v_j) \mid l_i < l_j < r_i < r_j\}$$

In other words, each vertex in the graph corresponds to an interval representing a net and a directed edge is drawn from $v_i$ to $v_j$ if and only if the interval defined by $N_i$ overlaps with that of $N_j$ but does not completely contains or is completely contained by $N_j$.

Similarly, define a containment graph $\bar{G}_C = (V, \bar{E}_C)$, where the vertex set $V$ is the same as defined above and $\bar{E}_C$ a set of directed edges defined below:

$$\bar{E}_C = \{(v_i, v_j) \mid l_i < l_j, r_i > r_j\}$$

In other words a directed edge is drawn from $v_i$ to $v_j$ if and only if the interval corresponding to $N_i$ completely contains the interval corresponding to $N_j$. Since

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
we will be dealing with both the directed and undirected versions of these graphs, the arrow head on top will be omitted to indicate the undirected graph. For example, by $G_C$ we mean the graph $\bar{G}_C$ ignoring the directions on the edges, i.e., $G_C = (V, E_C)$.

We also define an interval graph $G_I = (V, E_I)$ where the vertex set $V$ is the same as above, and two vertices are joined by an edge if and only if their corresponding intervals have a non-empty intersection. It is easy to see that $E_I = E_O \cup E_C$. Figure 2.1 shows an example of the overlap graph, the containment graph and the interval graph for a set of nets.

It is well known that the class of overlap graphs is equivalent to the class of circle graphs [3]. Similarly, the class of containment graphs is equivalent to the class of permutation graphs [14].
Figure 2.1 Graph Representation for a Net List.
CHAPTER III

MINIMUM-BEND SINGLE ROW ROUTING

The design of multilayer printed circuit boards (MPCB's) is of great importance in the design of complex electronic systems. So [32] proposed an approach for routing of MPCB's which decomposes the original problem into several smaller and simpler subproblems. One of the important subproblems is the single row (single layer) routing problem.

The single row routing problem has been extensively studied [2, 10, 17, 24, 34, 35, 36]. The problem has been shown to be NP-complete for many different objective functions, including minimization of tracks [24, 36] and minimization of doglegs [29], among others. The objective function most usually considered is minimization of maximum number of tracks on either side of the node axis and several heuristic algorithms have been proposed for this purpose [11, 34, 35]. Minimization of maximum number of tracks on either street is motivated by a need to minimize the overall area of the MPCB. The problem of minimizing maximum number of doglegs per net in a single row routing environment has been considered before [8]. In that paper, no algorithm was presented for routing, as the main focus of the authors was to develop theoretical results concerning the routability of SRRP when at most $K$ doglegs are allowed per net, for a fixed $K$.

In this chapter, we develop algorithms for single row routing problem which bound the maximum number of doglegs per net. Let us first briefly outline three applications which motivated our work:

1. In high-performance MPCB, the objective of single row routing problem is to minimize the length of the longest net. It is due to the fact that signal delay is proportional to the length of the longest wire. The unnecessary wire in a single row routing is mainly caused by the doglegs. For example the wire length of net

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
$N_2$ shown in Figure 3.1 (b) is much longer than the wire length of net $N_2$ shown in Figure 3.1 (a). Depending upon the track assignment, each dogleg increases the wire length of $N_2$ by some factor.

2. In the over-the-cell routing technique [17], each channel is collapsed into one set of terminals, and the area over the cells is used for routing. If we view the channel as the node axis, this problem is same as the single row routing problem. For this particular problem, each time a net is dogleged, the net needs to pass through the channel. Therefore, it is necessary to minimize the number of doglegs.

3. In fabrication of micro wave IC's [24], the traces in the metal layer do not act as simple electrical conductors; instead these paths act as waveguides. These traces are called microstrip lines. Each dogleg in a microstrip line results in reflections of the electromagnetic wave that makes the signal progressively weaker, and stronger drivers are thus required for proper transmission. Therefore, minimization of doglegs per net corresponds to optimization of transmission efficiency of microstrip lines.

In this chapter, we use a graph model for SRRP, which presents a given SRRP by an interval graph, an overlap graph and a containment graph. We develop three algorithms which use a graph decomposition technique to lay out a given SRRP and bound the maximum number of doglegs per net by the clique size of these graphs. The routing created by the first algorithm bounds the maximum number of doglegs per net by the clique size of the interval graph representing the given SRRP. We also present two more algorithms which improve upon this result and bound the maximum number of doglegs per net by the clique size of certain subgraphs. The clique size of an interval graph has an expected value of $2\sqrt{n} + 1$ [23], while in practical environments its value is $O(1)$. The time complexity of the second algorithm is $O(n^2)$, while other two algorithms run in $O(n \log n)$ time. Finally, we show that if a given SRRP has more than one connected component in the overlap graph, we can route each component separately and combine the
Figure 3.1 The Net Length and the Doglegs.
routings without any additional doglegs. The algorithms have been implemented in C on a SUN SPARCstation 1+ and experimental results show that for all practical examples, the proposed algorithms can produce a layout with at most two or three doglegs per net.

The chapter is organized as follows. In Section 1, we present three new algorithms for single row routing with bounded number of doglegs per net. Extension of the proposed algorithms is presented in Section 2. Experimental results are shown in Section 3. Finally, in Section 4, we give the summary.

3.1 Routing Based on Graph Decomposition

In this section, based on different graph decomposition techniques, we present three algorithms which route a given SRRP with bounded number of doglegs per net. The first algorithm is based on independent set decomposition, the second algorithm is based on maximum bipartite subset decomposition, while the third algorithm uses maximum non-containment subset decomposition.

3.1.1 Routing Based on Independent Set Decomposition

In this section, we present an algorithm which routes a given SRRP with at most $O(k)$ number of doglegs per net, where $k$ is the size of a maximum clique in the interval graph representing the SRRP. Expected value of $k$ is $\Theta(\sqrt{n})$ [23] and in practical examples $k$ is 6 or 7. The algorithm is based on the decomposition of the given SRRP into several smaller single row routing problems such that interval graph for each subproblem is null. We call this operation independent set decomposition of $G_I$. The motivation for this algorithm comes from the fact that using the interval graph representing a SRRP, the problem can be decomposed into $k$ subproblems and each one of these subproblems can be routed without any doglegs. Thus the key idea is to combine the routing of these subproblems such that maximum number of doglegs per net is minimized.
An independent set in an interval graph is a set of nets such that no two nets have containment or overlap relationship. We decompose a given SRRP \((L)\), with \(G_L\) denoting its interval graph, into \(k\) subproblems \((L_i; i = 1, \ldots, k)\) represented by subgraphs \(G'_i; i = 1, \ldots, k\) such that each pair of nets \(N_j, N_k\) of \(L_i\) satisfy the following condition:

\[ r_j > l_k \text{ or } r_k > l_j; j \neq k; \]

A set of nets which have a null interval graph is called an independent net list. The independent set decomposition of \(G_L\) can be done using the left-edge algorithm [16]. Using the \(k\) independent sets, we develop an algorithm, called \(k\)-dogleg-I(), which combines the routing of these sets into a routing for the given SRRP.

Let us first describe the algorithm informally. We label the \(k\) independent net lists produced by the left-edge algorithm as \(L_i; i = 1, \ldots, k\). The algorithm assigns the first independent net list \(L_1\) to the first track of the upper street and the second independent net list \(L_2\) to the first track of the lower street. It is clear that routing of these nets require no doglegs, as all nets in each set are independent. We insert the nets of the remaining \(k - 2\) independent net lists \((L_i; i = 3, \ldots, k)\) into the layout, one by one. While inserting an independent net list \(L_j\), we order the nets in \(L_j\) according to their left end points and insert them into the layout in a left to right fashion, starting with the leftmost net. (The ordering is crucial in proving the correctness of the algorithm.)

In order to find the optimal track assignment for a particular net \(N_i\), we consider the relationship of the net \(N_i\) with the nets which are already routed. It is clear that the net under consideration may overlap with some other nets and those nets have to be dogleged to "accommodate" \(N_i\). It is also obvious that net \(N_i\) causes unnecessary doglegs to nets which properly contain net \(N_i\), only if \(N_i\) is routed in a track "above" these nets. Let us assume that there exists a net \(N_j\) in the layout, which contains \(N_i\), that is \(l_j < l_i < r_i < r_j\). Furthermore, let us
assume that \( N_j \) is assigned to track \( t \). Now if \( N_i \) is assigned to a track \( t' > t \), it will result in four doglegs to \( N_j \). On the other hand, if \( N_i \) is assigned to track \( t' < t \), both nets can be routed without doglegs. This provides a simple method for finding an effective track assignment for the net under consideration. For a given net \( N_i \), we find a track numbered \( t \) such that all nets in tracks with number \( t' > t \) either contain \( N_i \) or do not overlap with \( N_i \). We assign \( N_i \) to track \( t \) and increase the track number assigned to each affected net by one. For example in Figure 3.2 (a), the net \( N_i \) is assigned to track 2 and all the nets (e.g., \( N_1, N_2, N_3 \)) in tracks above track 2 contain \( N_i \). The nets \( N_4, N_5, N_6 \) which are assigned to tracks below track 2, either intersect net \( N_i \) or are contained in it. Thus track 2 is the "optimum" track for net \( N_i \). During the insertion of a particular net, we simply connect the terminals of the net to the node axis, decompose each of the intersecting nets into three subnets. For example, in Figure 3.2 (b), net \( N_4 \) is decomposed into three subnets \( N_{4,1}, N_{4,2}, N_{4,3} \).

In order to minimize the number of doglegs, we alternate the insertion sequence of independent net lists between upper and lower streets. Thus, if \( L_i \) is inserted on the upper street, we insert \( L_{i+1} \) on the lower street. In this fashion at most \( \lceil \frac{k}{2} \rceil \) independent net lists are assigned to the upper street and \( \lfloor \frac{k}{2} \rfloor \) are assigned to the lower street.

We now present the formal description of the algorithm. For sake of completeness we include the independent set decomposition method using the left-edge algorithm.

Algorithm \( k\text{-dogleg-I()} \)

Input: A net list \( L \).
Output: A layout \( S \) of the given net list with at most \( O(k) \) doglegs per net.

\( T_p^t \): the \( p \)th track on the \( t \) street, where \( t \in \{upper, lower\} \);
Figure 3.2 The Insertion of Nets.
$U$: representing the upper street.
$B$: representing the lower street.

$u$: the number of tracks in the upper street.
$l$: the number of tracks in the lower street.

$Ledge(L)$: $k$ independent net lists obtained by left-edge algorithm.

Begin

**Phase 1:**

/* Use left-edge algorithm to get $k$ independent net lists

$(L_1, L_2, ..., L_k)$ of $L$. */

$L_i = Ledge(L); (i = 1, ..., k)$.

/* Assign $L_1$ to the upper street. */

FOR $(N_i \in L_1 \ i = 1, ..., m_1)$ \( T^U_1 = T^U_1 \cup N_i; \)

/* Assign $L_2$ to the lower street. */

FOR $(N_i \in L_2 \ i = 1, ..., m_2)$ \( T^B_1 = T^B_1 \cup N_i; \)

**Phase 2:**

/* Insert the remaining independent net lists. */

t = U; u = 1; l = 1;

FOR \( (G^i_j \ i = 3, ..., k) \)

FOR \( (N_j \in L_i (j = 1, ..., m_i)) \)

/* Find the smallest track which contains $N_j$. */

\[ k = \min\{q|1 \leq q \leq p, N_j \in T^i_q}\];

IF \( (N_j \text{ contained by previously routed net at } T^i_k) \)

/* Insert $N_j$ under $T^i_k$. */

insert($N_j$, $T^i_k$);
ELSE
   /* Assign the new net to the outer track. */
   \[ T_p^{n} = T_p \cup N_j; \]
   /* Switch street. */
   IF ( t = U )
      t = B; l = l + 1; p = l;
   ELSE
      t = U; u = u + 1; p = u;
End

In order to prove the bound on the maximum number of doglegs per net, we first investigate the effect of inserting one independent net list into an existing layout.

**Lemma 1** Given an existing routing \( E \) and an independent net list \( L_i \), inserting all the nets in \( L_i \) in the manner described will add at most add four more doglegs to each net in \( E \).

**Proof:** Let \( l_k \) and \( r_k \) denote the left and right terminal of the net \( N_k \) respectively. Given a net \( N_i \) to be inserted into an existing routing, and a net \( N_j \) in the existing routing, using the proposed insertion method, we have the following cases:

**Case 1:** \( (l_i < l_j < r_i < r_j) \) In this case, \( N_j \) intersects with \( N_i \), and needs to be dogleged. It is easy to see that \( N_j \) will have at most two more doglegs after inserting \( N_i \). For example, in Figure 3.3, \( N_a \) and \( N_1 \) are intersecting.

**Case 2:** \( (l_i < l_j < r_j < r_i) \) In this case, \( N_j \) is contained in \( N_i \). Thus \( N_i \) can be routed on a track \( t' > t \) (assume \( N_j \) is on track \( t \)), and thus \( N_i \) will not cause any doglegs to \( N_j \). In Figure 3.3, \( N_b \) and \( N_4 \) illustrate this case.

**Case 3:** \( (l_j < l_i < r_i < r_j) \) In this case, \( N_j \) contains \( N_i \). Let \( N_j \) be on track \( t \). We route \( N_i \) on a track \( t' < t \) and thus cause no additional doglegs to \( N_j \). Figure 3.3

---

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
shows an example of this case.

Case 4: \( l_j < l_i < r_j < r_i \) In this case, \( N_j \) intersects with \( N_i \) and \( N_j \) will have at most two more doglegs after inserting \( N_i \). In Figure 3.3, \( N_d \) and \( N_f \) are intersecting in this way.

Every two nets in one independent set do not intersect each other, and the nets which are contained by \( N_j \) or contain \( N_j \) will not dogleg \( N_j \). Therefore, after inserting one independent set, \( N_j \) will have at most four more doglegs as shown in Figure 3.3 (b). \( \Box \)

Thus inserting one independent set into an existing layout may cause at most \( O(1) \) doglegs to an already routed net.

**Corollary 1** Given an existing routing \( E \) with at most \( D_1 \) doglegs per net and \( k \) independent net lists \( L_i, (i = 1, 2, \ldots, k) \), after all the nets in \( L_i, (i = 1, 2, \ldots, k) \) have been inserted into \( E \), a net in \( E \) will have at most \( D_i + O(k) \) doglegs per net.

**Theorem 1** The algorithm \( k\text{-dogleg-I} \) can route a given net list \( L \) with at most \( O(k) \) doglegs per net in \( O(n \log n) \) time, where \( k = C_f \) and \( n \) is the total number of nets.

**Proof:** Given a net list \( L \), the algorithm decomposes it into \( k \) independent net lists \( L_i, i = 1, \ldots, k \), and routes the first independent net list \( L_1 \) on the upper street and the second independent set \( L_2 \) on the lower street. This operation can be done without any doglegs. Then the algorithm inserts all the nets in the remaining \( k - 2 \) independent net lists into the existing layout. According to Lemma 1, this insertion causes at most \( O(k) \) more doglegs to each net in the layout. So the total dogleg number per net is \( O(k) \). On the other hand, in an interval graph, \( k \) is equal to \( C_f \). Therefore, the algorithm \( k\text{-dogleg-I} \) can route a given net list with at most \( O(C_f) \) doglegs per net.
Figure 3.3 The Insertion of an Independent Set.
All operations of algorithm $k$-dogleg-I except finding $\min\{q|1 \leq q \leq p, N_j \in T_q\}$ can be carried out in constant time; each finding operation can be accomplished by a binary search in $O(\log n)$ time. Therefore the total time complexity is $O(n \log n)$. □

Although this algorithm has a good bound on number of doglegs per net, we can further improve this bound by using other graph decomposition methods. Two algorithms using these methods are proposed in the following sections.

3.1.2 Routing Based on Maximum Bipartite Subset Decomposition

The algorithm developed in the previous section has a performance which is bounded by the clique size of the entire graph representing a given SRRP. In this section, we develop a routing algorithm which bounds the maximum number of doglegs per net by the clique size of overlap subgraphs. By decomposing the overlap graph $G_0$ representing the SRRP, we produce a layout with a much smaller worst case bound. In this approach, we use the following fact (for more details see [8]):

**Theorem 2** A realization without doglegs is possible if and only if $G_0$ is a bipartite graph.

The basic idea of the algorithm is to decompose a given net list into two net lists $L_1$ and $L_2$ corresponding to the maximum bipartite subset (MBS) $G_0^1$ and $G_0^2$ in an overlap graph and a net list $L_3$ which contains all the remaining nets. In other words, we find a MBS of $G_0$ and $L_1$ and $L_2$ corresponding to the bipartite sets of MBS. Using Theorem 2, we route $L_1$ and $L_2$ without any doglegs, and insert all the remaining nets (i.e., $L_3$) into this layout. Since we can use independent set decomposition on $L_3$ and the insertion method of $k$-dogleg-I, the number of doglegs per net can be bounded by $O(C_3^1)$, where $C_3^1$ is the maximum clique size of $G_0^3$ representing $L_3$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Formally, we decompose a given net list $L$ into two net lists $L_1$ and $L_2$ represented by subgraphs $G^1_Q$ and $G^2_Q$ such that each pair of nets $N_j, N_k$ in the sublist satisfies:

$$r_j > l_k \text{ or } r_k > l_j, j \neq k;$$

or

$$l_j < l_k < r_k < r_j, j \neq k;$$

or

$$l_k < l_j < r_j < r_k, j \neq k;$$

However, finding the MBS in an overlap graph is NP-complete [26]. In order to find the MBS, we use the algorithm presented in [3], which finds two maximum independent sets (2-MIS), which is a $3/4$ approximation of the maximum bipartite subset in $O(n^2)$ time [17]. The algorithm of [27] to find a MBS in pseudo-exponential time or to find a MBS in optimal time for subclasses of the problem [22].

We route the two independent sets in 2-MIS, one on the upper street and one on the lower street. As the nets in any one set are pairwise non-overlapping, they can be routed without any doglegs. We now decompose remaining nets using independent set decomposition as described in the previous section and insert them into routing.

Formally, the algorithm is stated below:

Algorithm $k$-dogleg-II()

Input: A net list $L$.
Output: A layout $S$ of the given net list with at most $O(C^3_t)$ doglegs per net.

$T^t_p$: the $p^{th}$ track on the $t$ street, where $t \in \{upper, lower\};$
$U$: representing the upper street;
\[ B: \text{representing the lower street;} \]
\[ u: \text{the number of tracks in the upper street;} \]
\[ l: \text{the number of tracks in the lower street.} \]

Begin

Phase 1:

/* Use MIS algorithm to obtain bipartite sublists \( L_1 \) and \( L_2 \). */
/* and store all the remaining nets in sublist \( L_3 \). */
\[
L_1 = \text{MIS}(L);
L_2 = \text{MIS}(L - L_1);
L_3 = L - L_1 - L_2;
\]

/* Assign \( L_1 \) to the upper street. */
FOR ( \( N_i \in L_1 \) \( i = 1,..,m_1 \) ) \( T_p^U = T_p^U \cup N_i; \)

/* Assign \( L_2 \) to the lower street. */
FOR ( \( N_i \in L_2 \) \( i = 1,..,m_2 \) ) \( T_p^B = T_p^B \cup N_i; \)

Phase 2:

/* Use left-edge algorithm to get \( k \) independent sets
\((L_1, L_2,.., L_k)\) of \( L_3 \). */
\[ L_i = \text{Ledge}(L_3); (i = 1,..,k). \]

Phase 3:

/* Insert all the remaining nets in \( L_3 \). */
Same as Phase 2 of \( k \)-dogleg-I.

End
The worst case bound of this algorithm is better than \textit{k-dogleg-I} because removing MBS from the given net list reduces the density of the given problem considerably. In other words, maximum clique size in \(L_3\) is usually much smaller than \(L\).

**Theorem 3** The algorithm \textit{k-dogleg-II} routes a given instance of SRRP with at most \(O(C_f^3)\) doglegs per net in \(O(n^2)\) time, where \(C_f^3\) is the maximum clique size of the interval graph representing \(L_3\).

**Proof:** The algorithm first finds the two net lists corresponding to the maximum bipartite subset \(G_1^3\) and \(G_2^3\), assigns them one to the upper street and one to the lower street. So far no dogleg is needed according to Theorem 2. Then the algorithm decomposes the remaining nets into \(C_f^3\) independent net lists, and inserts them into the existing routing using Lemma 1. As this insertion will cause at most \(O(C_f^3)\) more doglegs to each net in existing layout, the entire routing is bounded by \(O(C_f^3)\) doglegs per net.

The time complexity of this algorithm is dominated by the algorithm \textit{MIS}, where the worst case running time it \(O(n^2)\) [3]. □

### 3.1.3 Routing Based on Maximum Non-Containment Subset Decomposition

In this section, we present another algorithm to route a given SRRP with bounded number of doglegs per net. The algorithm is motivated by the following fact, proof of which appears in [8].

**Theorem 4** If the containment graph \(G_C\) is null for a given SRRP then there exists a realization of SRRP with at most one dogleg per net.

Using Theorem 4, the algorithm finds a sub-net list \(L_1\), such that, containment graph of the nets in \(L_1\) is null. We call such a subset \textit{maximum non-containment subset} (MNS); it also called maximum proper subset in literature.
More precisely, the MNS is a subset of a given set of nets in which no two nets have containment relationship. In other words, all the nets in this subset satisfy the following:

\[ l_j \leq l_k < r_j < r_k \text{ or } r_j < l_k, j \neq k; \]

or

\[ l_k < l_j < r_k < r_j \text{ or } r_k < l_j, j \neq k; \]

We route \( L_1 \) with at most one dogleg per net and use independent set decomposition routing scheme as described in Section 3.1 for the remaining nets \( (L_2 = L - L_1) \). In this way, the algorithm creates a layout in which the number of doglegs per net is bounded by \( O(C^2) \), the size of the maximum clique in the interval graph representing nets in \( L_2 \). We start with developing an algorithm for MNS. First, we describe the basic idea informally, followed by a detailed algorithm.

Using interval representation of a net list \( L \), we renumber the nets from left to right by their left terminal positions \( (L = \{N_1, N_2, \ldots, N_n\}) \). With respect to this new numbering, we define several sub-net lists, \( L_i = \{N_1, N_2, \ldots, N_i\} \), \( 1 \leq i \leq n \). Note \( L_{i+1} = L_i \cup N_{i+1} \). Let us further assume that the size of the MNS is \( M \). The basic idea of this algorithm is to inductively compute MNS from \( L_1 \) to \( L_n \). Starting with an empty set, in \( i^{th} \) iteration of the algorithm, we compute MNS for \( L_i \). MNS of \( L_i \) is computed using the MNS of all \( L_j, 1 \leq j \leq i \). For each size \( m, 1 \leq m \leq M \), we keep track of only one feasible solution instead of many feasible ones. We show that keeping one solution is sufficient because we choose the solution that permits us to find a larger solution later if one exists. For each \( m, 1 \leq m \leq M \), we represent the MNS by its rightmost net and a link to the partial solution of the previous step. In this way, the rightmost nets of the solution are ordered from left to right with increasing size of solution, and the algorithm ends with the MNS of the entire net list. The procedure of MNS is illustrated in Figure 3.4.

The formal algorithm is given as follows:
Figure 3.4 Illustration of Algorithm find-MNS.
Algorithm \textit{find-MNS()} \\

Input: A net list \(L\).
Output: An array \(S\) which contains MNS of \(L\).

\(T(i)\): An array which contains the last nets of MNS whose size is from 1 to \(i\).

Begin
\[
T(0) = 0; \\
last = 0; \\
\text{FOR } N_i \text{ to } N_n \text{ DO} \\
\quad \text{IF } r_{T(last)} < r_i \quad /* r_i \text{ is the right terminal of net } i. */ \\
\quad \quad \text{THEN } last = last + 1; k = last; \\
\quad \quad \text{ELSE } k = \min\{j|1 \leq j \leq last, r_{T(j)} > r_i\}; \\
\quad \quad T(k) = i; \\
\quad \quad R(i) = T(k-1); \\
\quad /* solution is found: trace it back following the links. */ \\
\quad \text{FOR (last - 1) to 1 DO} \\
\quad \quad S(k) = R(T(k+1)) \\
\quad /* solution is stored in } S. */ \\
\text{End}
\]

Now we give a formal proof of correctness of our algorithm, and analyze its time and space complexities. In the algorithm, \(T\) is an array of pointers such that \(T(i)\) points to the last net in MNS \(S(i)\).

\textbf{Lemma 2} \(T(last + 1) = N_{i+1}\), if and only if \(r_{i+1} > r_{T(last)}\).
**Proof:** For the if part, note \( r_{i+1} > r_{T(last)} \). This implies that \( N_{i+1} \) is not contained by any net in \( T(last) \). In other words, \( N_{i+1} \) can be added into the current MNS \( S(last) \) to make it larger, therefore \( T(last + 1) = N_{i+1} \).

For the only if part, we can see that if \( T(last + 1) = N_{i+1} \), then \( N_{i+1} \) is not contained by any net in \( T(last) \), therefore \( N_{i+1} \) is not contained by the last net in \( T(last) \). Also note \( l_{i+1} > l_{T(last)} \), therefore \( r_{i+1} > r_{T(last)} \). □

**Lemma 3** *In the algorithm, if \( T(j) \) points to the last net of \( S(j) \) in the \( i^{th} \) inductive step, then in the \( (i + 1)^{th} \) inductive step, it still points to the last nets of \( S(j) \), \( 1 \leq j \leq last \).*

**Proof:** In the algorithm find-MNS, for each inductive step we have the following two cases:

**Case 1:** \( (r_{T(last)} < r_{i+1}) \). In this case, according to Lemma 2, the net \( N_{i+1} \) can be added into the current MNS set. \( T(last + 1) \) points to the last net \( N_{i+1} \) in \( S(last + 1) \).

**Case 2:** \( (r_{T(last)} \geq r_{i+1}) \). In this case, the size of MNS will not increase. But it is possible that the new net \( N_{i+1} \) may lead to a better solution with the nets which start later. Therefore, the algorithm finds the first net \( N_j \) which contains the new net \( N_{i+1} \), and replaces \( N_j \) with \( N_{i+1} \). At this point, note that \( N_j \) contains \( N_{i+1} \), implying that this replacement will not reduce the size of MNS consisting of \( N_j \), as all the nets which are not contained by \( N_j \) are also not contained by \( N_{i+1} \). Therefore, all the last nets for all possible size MNS solutions are kept in the array \( T \) after this operation. □

**Theorem 5** *The algorithm find-MNS finds the maximum non-containment subset in \( O(n \log s) \) time with \( O(n) \) space.*

**Proof:** To prove the correctness of the algorithm, we use induction on step \( i \).

**Base Case:** note that the algorithm considers the first net and assigns it to the array \( T(1) \). Obviously, the current MNS \( S(1) \) can be obtained from \( T(1) \).
**Inductive Hypothesis:** Assume that in step $i$, the algorithm has found the current MNS, we need to prove that in step $i + 1$, the algorithm still can find the MNS.

**Inductive Step:** If in step $i$ the algorithm has updated the array $T$ and $T$ points to the last net of current MNS $S$, then using Lemma 3, in step $i + 1$, the algorithm still can update the array $T$ such that $T$ still points to the last nets of $S$.

Therefore, the array $T$ points to all the last nets of all the possible size solution until the last net $N_n$ is considered. The best solution can be traced back using $T$ and the pointers in the MNS chain.

For the time and space complexity, we can see that all operations, except finding $\min\{j|1 \leq j \leq \text{last}, r_{T(j)} > r_i\}$, can be carried out in constant time; the $\min$ operation can be implemented using binary search in $O(\log s)$ time, as we only need to search the array $T$. Depending on the cases in Lemma 3, $s$ nets can be directly added to the MNS in constant time, while the other $n - s$ nets are added using a binary search. Therefore the total time complexity is $O(s + (n - s)\log s) = O(n \log s)$.

The solution is kept in the array $T$ which needs $O(s)$ space. In the worst case, we need $O(n)$ pointers, so the space complexity is $O(n)$.□

Now we present the algorithm $k$-dogleg-III. The basis of this algorithm is to use algorithm find-MNS to find the maximum non-containment subset and route those nets with one dogleg per net. Then the algorithm inserts all the remaining nets into the routing. The formal algorithm is as follows:

**Algorithm $k$-dogleg-III()**

Input: A net list $L$.

Output: A layout $S$ of the given net list with at most $O(C_1^2)$ doglegs per net.
$T_p^t$: the $p^{th}$ track on the $t$ street, where $t \in \{\text{upper, lower}\}$; 

$U$: representing the upper street; 

$B$: representing the lower street; 

$u$: the number of tracks in the upper street; 

$l$: the number of tracks in the lower street.

Begin

**Phase 1:**

/* Use \textit{find-MNS} algorithm to get the maximum overlap subset $L_1$ */
/* and store all the remaining nets in net list $L_2$. */

$L_1 = \text{find-MNS}(L)$;

$L_2 = L - L_1$;

/* Assign $L_1$ on the upper street and lower street. */

FOR ( $N_i \in L_1$ $i = 1,..,m_1$ )

$T_p^U = T_p^U \cup N_i$;

$T_p^B = T_p^B \cup N_i$;

**Phase 2:**

/* Use \textit{left-edge} algorithm to get $k$ independent sets */
/* $(L_1, L_2,.., L_k)$ of $L_2$. */

$L_i = \text{Ledge}(L_2); (i = 1,.., k)$.

**Phase 3:**

/* Insert all the remaining nets in $L_2$. */

Same as Phase 2 of \textit{k-dogleg-I}.

End

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The worst case bound on the number of doglegs per net of this algorithm is based on the independent set decomposition of \( L_2 \).

**Theorem 6** The algorithm \( k\)-dogleg-III can route a given SRRP with at most \( O(C^2_l) \) doglegs per net in \( O(n \log n) \) time, where \( C^2_l \) is the maximum clique size of \( L_2 \).

**Proof:** The given net list is decomposed into two net lists corresponding to \( G^1_o \) and \( G^2_o \), where \( G^1_o \) is the maximum non-containment subset and \( G^2_o \) is the set containing all the remaining nets. According to Theorem 4, a routing of \( L_1 \) can be obtained with one dogleg per net. \( L_2 \) can be partitioned into \( C^2_l \) independent net lists, and be inserted into the routing with \( O(C^2_l) \) more doglegs per net. Therefore, the algorithm \( k\)-dogleg-III can route a given SRRP with at most \( O(C^2_l) \) doglegs per net.

As the time complexity of Phase 1 is \( O(n \log n) \), Phase 2 is \( O(n) \) and Phase 3 is \( O(n \log n) \), the time complexity of algorithm \( k\)-dogleg-III is \( O(n \log n) \). □

### 3.2 Extension of the Proposed Algorithms

In this section, we show that performance of the proposed algorithms can be further improved by using the decomposition of a given SRRP based on connected components of the overlap graph representing the problem. We show that if \( G_o \) has more than one connected component, then each component can be routed separately and routings of all the components can be combined without any additional doglegs.

It is easy to see that for a given SRRP if \( G_o \) contains more than one connected component then these components are related in a "tree like fashion." To explore the relationship among the connected components of \( G_o \) we define a directed graph \( T_i \) as follows: Let \( H = \{H_i \mid H_i = (V_i, E_i), 1 \leq i \leq r\} \) be the set...
of connected components of $G_0$. We define a directed graph $\overline{T} = (V^h, \overline{E}^h)$, where $V^h = \{v^h_1, v^h_2, \ldots, v^h_r\}$ such that there exist a bijection from $H$ to $V^h$.

The edge set $\overline{E}^h$ is defined as follows:

$$\overline{E}^h = \{(v^h_i, v^h_j) \mid \exists u \in V, \exists v \in V_j, (u,v) \in \overline{E}_C, 1 \leq i, j \leq r\}$$

In other words, a directed edge is drawn from $v^h_i$ to $v^h_j$ if and only if the composite interval $CI_{H_i}$ corresponding to $H_j$ is completely contained in the composite interval $CI_{H_i}$ corresponding to $H_i$, where composite interval of a connected component $H_i$ is defined as $CI_{H_i} = (\min_k l_k, \max_k r_k)$ for all $k, N_k \in H_i$. A directed graph $\overline{G} = (V, \overline{E})$ is called transitive if for every pair of edges $e_1 = (u_1, u_2)$ and $e_2 = (u_2, u_3)$ there exist an edge $e_3 = (u_1, u_3)$.

First we present the algorithm $m$-dogleg followed by the proof of correctness.

Algorithm $m$-dogleg()

Input: A connected component tree $\overline{T}_i$.

Output: Permutation of the given Net list with at most $M$ doglegs per net.

$P_i$: The permutation of node $i$.

Begin

While not empty($\overline{T}_i$) do

1 Find a node $H_i$ whose children have zero out-degree;

2 For each Child $H_{ij}$ of $H_i$ do

2.1 Find a valid insertion position $t$ in $P_i$;

2.2 Insert $P_{H_{ij}}$ into $P_i$;

3 Delete each Child $H_{ij}$ of $H_i$;

4 If $H_i$ is root then delete $H_i$;

End
Theorem 7 Let $G^i_0, i = 1, \ldots, m$, be the connected components of $G_0$. If $G^i_0$ is routable with $k_i$ doglegs per net, then the algorithm $m$-dogleg() can produce a routing of the given SRRP with $M$ doglegs per net in $O(n)$ time, where $M = \max_{i=1}^{m} k_i$.

Proof: If $G_0$ has only one connected component, the theorem is obviously true. Therefore, we assume that $G_0$ has two or more connected components. Moreover, we assume that each connected component $G^i_0$ can be routed using at most $M$ doglegs per net. The problem is thus reduced to optimal composition of connected components such that no net is doglegged more than $M$ times, where $M = \max_{i=1}^{m} k_i$.

The connected components are optimally composed using $\hat{T}_i$. Intuitively speaking, $\hat{T}_i$ shows the containment relationship between connected components of $G_0$. Thus tree $\hat{T}_i$ can be used to optimally compose the routing by inserting the routing of a component into the existing routing. In each insertion, we use the parent tree $\bar{T}_i$ to find a node $P$ such that all its children have zero out-degree in $\bar{T}_i$. We then insert the routing of each child into the routing of its parent node $P$. During the insertion, we simply assign the child routing "inside" the parent routing as shown in Figure 3.5. We call this type of insertion a valid insertion.

After the insertion of the routing of a child into the routing of its parent, the algorithm deletes the child node. The algorithm stops after the insertion step in which the root node is deleted. We can see that valid insertion position always exists as no parent net can start or end in the composite interval of any of its children, and this insertion does not change the number of doglegs per net of any connected components.

For each connected component, we need $O(n)$ time to insert it, as we have $d$ connected components, therefore, we need $O(dn)$ time to finish the entire routing.

$\Box$
Figure 3.5 An Example of a Valid Insertion.
For practical routing problems, the approach presented above is very useful because most practical examples are composed of several connected components.

3.3 Experimental Results

The proposed algorithms have been implemented in C on a Sun SPARCstation 1+. For comparative purposes, we have also implemented the *left-edge* algorithm. The proposed algorithms perform very well on randomly generated examples and usually produce a layout with at most two or three doglegs per net. In practice, a single row routing problem may have the maximum clique size equal to five or six. Thus, our first algorithm may create a layout with three to four doglegs per net, while the other two algorithms may produce the routing with at most one to two doglegs per net.

Using normal distribution, we generated 50 test examples ranging in size from 10 to 50 nets. Experimental results show our algorithms produce much better results than *left-edge* algorithm and the worse case bounds proved in earlier sections.

For most instances, algorithms *k-dogleg-II* and *k-dogleg-III* obtain better results than algorithm *k-dogleg-I*. This is due to the fact that the algorithms *k-dogleg-I*s and *k-dogleg-II*s performance is bounded by the maximum clique size of a subgraph representing the given SRRP, while the performance of the first algorithm is bounded by the maximum clique size of the entire graph representing the given SRRP. For the instances in which $C_f^2$ is smaller than $C_f^3$, algorithm *k-dogleg-III* produces better results than algorithm *k-dogleg-II*. On the other hand, for the instances in which $C_f^2$ is bigger than $C_f^3$, algorithm *k-dogleg-II* gets better results than the algorithm *k-dogleg-III*. Therefore, one may wish to compute $C_f^2$ and $C_f^3$ to decide which algorithm should be used. Figure 3.6 shows the actual layouts created by the four algorithms in which the algorithm *k-dogleg-II* produces the best layout.
Figure 3.6 A Layout Instance Created by the Four Algorithms.
Another key feature of the proposed algorithms is their fast running times. Table 3.2 shows the average running times of all the algorithms on the test data. It can be seen from the table that the proposed algorithms have running times which are very comparable to that of the Left Edge algorithm.

We have analyzed the test results in terms of various parameters. First, we consider the distribution of doglegs among nets. Ideally we would like to minimize the number of nets with large number of doglegs. Figure 3.7 shows the dogleg distribution for \textit{k-dogleg-II}. It shows that most nets have very small number of doglegs, in fact, few nets have more than 3 doglegs. Figure 3.8 shows the relationship between clique size and maximum number of doglegs per net. It can be seen that the maximum number of doglegs per net for the proposed algorithm is much smaller than the \textit{left-edge} algorithm and grows slowly as a function of clique size. Last, we studied the effect of number of nets in a problem instance. The relationship between problem size and maximum number of doglegs per net is shown in Figure 3.9. The figure also shows the clique size and compares our results with the \textit{left-edge} algorithm. It is clear that our algorithms produce layouts with numbers of doglegs per net in the order of the clique size.

3.4 Summary

In this chapter, we have presented three algorithms for minimum-bend single row routing problem. The proposed algorithms have a performance which is bounded by the clique size in certain subgraphs. This performance bound leads to extremely efficient routing as the maximum clique size in SRRP is usually small.

The significance of our results lies in the fact that all previously proposed algorithms have no performance bound. We bound the performance very tight and showed experimentally that bounds hold for randomly generated test cases. Another reason as to why we get a performance bound on the routing is due to our
Table 3.1
Experimental Results: The Average Maximum Number of Doglegs Per Net

<table>
<thead>
<tr>
<th>Nets</th>
<th>k-dogleg-I</th>
<th>k-dogleg-II</th>
<th>k-dogleg-III</th>
<th>left-edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>C_I</td>
<td>max_leg</td>
<td>C_II</td>
<td>max_leg</td>
</tr>
<tr>
<td>10</td>
<td>7</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>15</td>
<td>12</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>24</td>
<td>18</td>
<td>13</td>
<td>10</td>
</tr>
<tr>
<td>40</td>
<td>29</td>
<td>20</td>
<td>21</td>
<td>14</td>
</tr>
<tr>
<td>50</td>
<td>32</td>
<td>24</td>
<td>22</td>
<td>15</td>
</tr>
</tbody>
</table>

max_leg: the average maximum number of doglegs per net.
(For each set of nets n, 10 examples were tested.)

Table 3.2
Experimental Results: Running Time

<table>
<thead>
<tr>
<th>Nets</th>
<th>k-dogleg-I</th>
<th>k-dogleg-II</th>
<th>k-dogleg-III</th>
<th>left-edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>Sec.</td>
<td>Sec.</td>
<td>Sec.</td>
<td>Sec.</td>
</tr>
<tr>
<td>10</td>
<td>0.75</td>
<td>0.70</td>
<td>0.65</td>
<td>0.70</td>
</tr>
<tr>
<td>20</td>
<td>0.70</td>
<td>0.75</td>
<td>0.75</td>
<td>0.80</td>
</tr>
<tr>
<td>30</td>
<td>0.85</td>
<td>0.80</td>
<td>0.80</td>
<td>0.80</td>
</tr>
<tr>
<td>40</td>
<td>0.95</td>
<td>0.95</td>
<td>1.00</td>
<td>0.90</td>
</tr>
<tr>
<td>50</td>
<td>1.15</td>
<td>1.10</td>
<td>1.05</td>
<td>0.95</td>
</tr>
</tbody>
</table>

For each set of nets n, 10 examples were tested.
Figure 3.7 Dogleg Distribution of Nets.
Figure 3.8 Relationship Between Maximum Number of Doglegs Per Net and Clique Size.
Figure 3.9  Relationship Between Maximum Number of Doglegs Per Net and Problem Size.
global perspective. Graph representation allows us to consider global relationship between nets and leads to better routings. Existing algorithms usually proceed in a greedy, left-to-right, fashion and therefore are unable to minimize the maximum number of doglegs. We believe that our algorithm will be useful in development of a practical router for high performance multilayer PCBs, over-the-cell routing and micro wave IC's.
CHAPTER IV

OVER-THE-CELL ROUTING BASED ON NEW CELL MODEL

Recently, the standard cell design style has increased in popularity as it offers a good compromise between time-to-market and area. In other words, standard cell offers a fast turnaround time without poor real estate utilization as in completely synthesized FPGAs and other gate array technologies. Therefore, standard cell is playing an important role in the market, and this role is likely to expand in the future.

In order to be more competitive with full-custom designs, area-efficient standard cell design technologies must be developed. The most common means of reducing area in standard cell layouts is to eliminate some of the channel routing area in the design. This problem has been extensively studied, and many two layer and three layer channel routers have been developed, several of which can produce "near optimal" routings for most channels. (Refer to [20], Chapter 9.) Despite this fact, as much as 15% of the area in most standard cell layouts is still consumed by channel routing. As a result, several researchers have investigated the use of area over the cells to obtain further reduction in channel height [5, 6, 15, 17, 18, 21, 30]. This routing technique is referred to as over-the-cell routing and is possible due to the fact that most cell libraries do not allow use of the second (M2) and third (M3) metal layers for connections within the cells.

Over-the-cell channel routing is a superproblem of the traditional channel routing problem; hence, like channel routing, it is NP-hard [33]. Three heuristics for over-the-cell channel routing have been presented for the two metal layer process [5, 17, 21]. Two of them consider only nets whose terminals are on a single row, either top or bottom, and produce up to a 20% reduction in channel height for over-the-cell channel routing, as compared to conventional channel
The third router allows a much wider variety of nets to be routed over the cell rows and produces up to a 35% reduction in channel height as compared to conventional channel routers [17]. Two heuristics have been presented for over-the-cell routing using the three metal layer process, both of which produce a 59% reduction in channel height as compared to a traditional three layer channel router [17].

Despite these excellent results, all existing over-the-cell routers suffer from one major drawback. They use a cell layout style (cell model) which assumes that terminals are available in all layers on the cell boundary. This cell model, while feasible for two layer routing, does not correspond to the new three layer cell model currently under development in industry, as most industrial models provide terminals in the second metal layer (M2) in the middle of the cell. This difference between existing and new cell models makes adaptation of existing over-the-cell routers to suit the new cell model difficult.

In this chapter, we develop a new cell model and a corresponding over-the-cell routing algorithm (ICR-3). Our cell model is representative of the three layer cell libraries currently under development in industry, and hence the ICR-3 router is suitable for industrial designs. In our routing algorithm, nets are partitioned into two sets. The nets in first set are called critical nets and are routed in the channel using direct vertical wires on the M2 layer. Critical nets are selected based on weight, where the weight of a net indicates the reduction in the channel height possible if this net can be eliminated from the channel. The actual selection process is performed by defining a vertex-weighted permutation graph for the nets in the channel and selecting the nets which correspond to a maximum-weighted independent set in that graph. The critical nets partition the M2 routing area into several regions. The remaining nets are assigned terminal positions on the boundary of the cell row and are routed to their terminal positions on M2 in the regions defined by the critical nets. Since we have choice of assigning terminals to nets on the boundary, we permute the nets such that all vertical constraints are
removed, and the congestion in the resulting channel is minimized. We present a $O(ks \log s)$ heuristic to solve this terminal permutation problem, where $k$ is the number of regions and $s$ is the maximum number of nets in a region. It should be noted that the terminal permutation complete defines the channel routing problem. We then select a planar subset of nets to route over the cells on the third metal (M3) layer such that the density of the nets remaining in the channel is minimum. The nets in the channel are routed using an HVH router.

We have implemented our router in C on a SUN SPARCstation 1+ and tested it on several benchmarks including PRIMARY I and PRIMARY II from MCNC. We have also implemented some other existing routers for comparison purposes. Both of our routers out perform all existing routers. We show that our three layer router (ICR-3) produces results which are better (on the average) by 58% as compared two layer over-the-cell (2OTC) router and 47% as compared to a conventional three layer channel (3CRP) router. ICR-3 routes the entire Primary I using only 104 tracks, which is a 40% better than a 3CRP router and 51% better than best known 2OTC router. In addition, this routing of PRIMARY I has 56% fewer vias as compared to 3CRP router. Furthermore, the ICR-3 is very fast, for example, it completes the routing of entire PRIMARY I benchmark in 5.8 seconds.

The chapter is organized as follows. In Section 1, we describe the details of the cell model used by existing over-the-cell routers and present a new cell model similar to those currently under development in industry. Section 2 provides an overview of the ICR-3 router, and Section 3 discusses details of algorithm ICR-3. In Section 4, a formal statement of the algorithm is presented. Sections 5 and 6 contain experimental results and summary, respectively.

4.1 The Existing and the New Cell Models

In this section, we describe the differences between the existing and the new cell model and their effect on routing techniques.
4.1.1 Existing Model

In the traditional cell layout style, there are two parallel horizontal diffusion rows, one for the P-type transistors and the other for N-type transistors. The first metal layer (M1) is used to complete connections which are internal to the cells. Feedthrough routing is also done using the M1 layer. Power and ground lines are routed in the second metal layer (M2) in the center of the area over-the-cell rows. Terminal rows are available in all layers and are located on the boundaries of the cells [17]. This leaves a rectangular, over-the-cell routing area for each terminal row of the standard cells. The number of tracks available for over-the-cell routing is determined by the height of these rectangular areas and may vary depending on the cell library used. Cell height is usually assumed to be 150\(\lambda\), which leaves thirteen tracks available in each rectangle. It should be noted that the entire over-the-cell area may be used for routing in the third metal (M3) layer. The M3 area is partitioned horizontally into two rectangular regions of equal size. One region is used for over-the-cell routing in the upper channel (above the cell row) and the other is allocated to the lower channel (below the cell row). If we use the 150\(\lambda\) cell library, thirteen tracks are available for routing in the M3 layer of the over-the-cell regions. This model is used by most existing over-the-cell routers [6, 17, 21].

4.1.2 New Cell Model

The cell model currently being adopted by the industry for the three metal layer process is quite different in terms of terminal locations than the existing cell model described above. In the new model, the terminals are located in M2 in the middle of the cell. The power and ground rails are in M1 near the top and bottom cell boundaries respectively. The connections within the cell are completed in M1. Thus, M2 is only blocked by terminals, and M3 is completely unblocked. Over-the-cell routers may use two rectangular regions (about thirteen tracks wide) in
M2 and M3. Vias may or may not be allowed in over-the-cells areas depending upon the process.

The key differences between this model and the existing model are the location of terminals and the assumption of availability of terminals in all metal layers. We observe that these differences make it very difficult to adapt existing over-the-cell routers to new cell models. Examples of each of these models are shown in Figure 4.1.

4.2 An Overview of ICR-3

In this section, we present an overview of our three layer over-the-cell router. In three metal layer environment, M2 and M3 layers are available for routing over the cells, as it is assumed that M1 is used for routing within each cell. Although two layers are available for routing, their use is somewhat restricted due to restriction on the use of vias. We do not use vias in the over-the-cell (OTC) area, as many technologies forbid use of vias over cells and active elements due to fabrication problems. As a result, the routing in OTC areas on both layers must be planar. As a result, the terminals cannot be "brought up" to M3, as that would require a via, and wires from all the terminals must be routed on M2. Vias may be used to complete the connections in the channel. This planarity condition is the most important consideration in development of OTC routing algorithm for the new cell model. The basic steps of the ICR-3 algorithm are as follows and a routing example is given in Figure 4.2.

1. **Net Classification:** In this step, all the nets are classified into three types. *TYPE I nets* are the nets whose terminals are on a same terminal row. *TYPE II nets* are two terminal nets whose terminals are on different terminal rows (One on the top terminal row and one on the bottom terminal row). *TYPE III nets* are multi-terminal nets whose terminals are on both top and bottom terminal rows.
Figure 4.1 Cell Models.
Number of Tracks: 1; Number of Vias: 12.

Figure 4.2 A Routing Example Created by ICR-3.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
2. Net Weighting and Net Selection for M2 Layer: In this step, all TYPE II nets are assigned weights. The weight of a net is based on several factors, such as, criticality of the net, its contribution to channel congestion, among others. Based on the weights assigned to the nets, we find a maximum independent set of the nets. And all the nets in this set are routed in M2 and pass through the channel as a straight wire. These nets do not contribute to channel density, and partition M2 into several regions.

3. Decomposition of Multi-Terminal Nets: After obtaining the initial regions, we partition the multi-terminal nets which have terminals on both top and bottom terminal rows (TYPE III nets). The objective of this operation is to increase the number of nets which can be routed straight in the channel on M2.

4. Terminal Assignments of Critical Nets: After the routing regions in M2 is topologically fixed, we assign the geometric terminal positions to the critical nets, to obtain the geometric definitions of the regions. This is done to maximize the routability of the nets in the M2, as well as to satisfy the congestion requirement of each region. We develop an optimal algorithm for this problem based on dynamic programming.

5. Terminal Assignments of Non-Critical Nets: Within each region, we assign the terminal positions to the remaining nets, to achieve three objectives: First, to eliminate vertical constraints, second, to minimize horizontal constraints and third to maximize the MBS in a special graph called overlap graph defined by the intervals of the nets.

6. M2 River Routing: After all the terminal positions are fixed, we use a river router to route the nets in a planar fashion in M2 OTC area.

7. Net Selection for M3 Layer: Among the remaining nets, we select a set of nets which forms a MBS in a circle graph. Since this problem is known to be NP-Hard, we use an approximate algorithm.
8. **M3 OTC Routing**: In this step, the nets selected in the previous step are routed in a planar fashion in M3 OTC areas on both sides of the channel. This step is similar to existing OTC routers.

9. **Channel Routing**: We use a HVH routing model to route the remaining nets in the channel. It must be noted that since all vertical constraints are eliminated, we use a router based on *left-edge* algorithm, adapted for three layer routing environment.

10. **Clean up Routing**: Finally, we consider all multi-terminal nets and re-compose the net as much all possible. This step removes the redundant wiring and improves the layout.

### 4.3 Detailed Description of the Algorithm

In this section, we present details of various phases of the algorithm ICR-3. The formal statement of the algorithm is given in Section 6.

#### 4.3.1 Net Classification

The first step in the algorithm ICR-3 is net classification. In this step, all the nets are classified into three types, based on the number of terminals of the net and the row to which each terminal belongs:

1. **TYPE I nets**: The nets whose terminals are on a same terminal row.
2. **TYPE II nets**: The two terminal nets whose terminals are on both top and bottom terminal rows.
3. **TYPE III nets**: The multi-terminal nets whose terminals are on both top and bottom terminal rows.

This classification operation is rather straightforward and can done in $O(T + B)$ time, where $T$ is the number of terminals in the top row and $B$ is the number of terminals in the bottom row of the channel.
4.3.2 Net Weights and Selection for M2 Layer

The purpose of this phase is to assign a weight to each net, which indicates the importance of that net for over-the-cell routing. Then we select a subset of nets, which are routed in a straight manner on M2 in OTC and channel area and are used to partition the routing area into several smaller routing regions.

In order to simplify the overall routing problem, we first consider those two terminal nets which have one terminal on the top terminal row and one terminal on the bottom terminal row, i.e., TYPE II nets.

Given a two terminal net $n_i = (t_i, b_i)$, where $t_i$ and $b_i$ denote the top and bottom terminals of the net. The weight of a net $n_i$ is denoted by $wt(n_i)$ and is computed as follows:

$$wt(n_i) = f_1 \times Crit(n_i) + f_2 \times len(n_i) + f_3 \times max\_chain(VCG, n_i) + f_4 \times max\_density(HCG, n_i)$$

The weight of a net consists of four factors. The first factor considers the timing requirement, which may force the layout tool to route a net with minimum wire, thus $Crit(n_i)$ is a number assigned to each net as per its timing requirement, higher number for a net indicates higher criticality of that net’s timing. In order to have rectangular regions, we would like to route the selected net as “straight” as possible. As a result, the second factor considered, is the length of the net. The shorter nets are assigned higher values than the longer nets. The remaining two factors take into account a net’s contribution to channel density. The term $max\_chain$ is computed by finding the ancestor and descendent weights of a net in the vertical constraint graph. Intuitively, it indicates the position of a net in a maximum chain in the vertical constraint graph. A net which is in the middle of a vertical constraint chain and therefore its deletion may cause a significant reduction in the height of the VCG, is given a high $max\_chain$ value. Similarly, $max\_density$ is simply the size of the maximum clique in horizontal constraint.
graph, to which a net belongs. The multiplicative factors $f_1$, $f_2$, $f_3$ and $f_4$ are experimentally determined and are fine-tuned during experimentation.

After computing the weight of each net, we select the nets which are to be routed straight in the channel on M2, based on a special graph called permutation graph. Let us consider a channel routing problem consisting of two terminals nets and defined by a top and a bottom row of terminals. Let $T = (t_1, t_2, \ldots, t_n)$ be the top row of terminals and $B = (b_1, b_2, \ldots, b_n)$ be the bottom row of terminals. Let $N = \{n_1, n_2, \ldots, n_m\}$ be the set of two terminal nets. We draw a visual representation of the problem by drawing a line between the terminals of the net. We define a permutation graph $G = (V, E)$, as follows:

$V = \{v_i| \text{vertex } v_i \text{ corresponds to net } n_i\}$

and

$E = \{(v_i, v_j)| \text{if lines representing nets } n_i \text{ and } n_j \text{ intersect}\}$

A permutation graph for the given channel in Figure 4.2 is shown in Figure 4.3. The weight of a vertex $v_i \in V$ is simply the weight of net $n_i$.

Our objective is to select a maximum set of non-intersecting nets, that is, a set of nets that can be routed in a planar fashion on M2. In addition, we would also like to select the set with maximum weight. It is easy to see that this problem corresponds to finding the maximum weighted independent set problem in a permutation graph $G$. This problem can be solved very efficiently by the algorithm presented in [37], which has the worst case time complexity of $O(n \log n)$.

The selected nets are routed straight in the channel on M2 and partition the M2 routing area (Channel and OTC area) into several regions as shown in Figure 4.3.
(A) A Routing Example.

(B) The Permutation Graph.

Figure 4.3 An Example of the Permutation Graph.
4.3.3 Decomposition of Multi-Terminal Nets

In this phase of the algorithm, we partition the multi-terminal nets which have terminals on both top and bottom terminal rows (TYPE III nets), based on the regions defined in the previous step. The objective of this operation is to increase the number of nets which can be routed straight in the channel on M2.

To preserve the routing regions obtained in the previous step, we solve this problem on a region by region basis.

Let \( t_{ij}^1, t_{ij}^2, \ldots, t_{ij}^p \) be the set of top terminals of net \( n_i \) in region \( R_j \). Similarly, let \( b_{ij}^1, b_{ij}^2, \ldots, b_{ij}^q \) be the set of bottom terminals of net \( n_i \) in region \( R_j \). We define decompose each net into several two terminal nets \( n_{ik} = \{t_{ik}, t_{ik}^i \} \) for \( k = 1, \ldots, \min(p, q) \). If \( p \neq q \) then the remaining terminals are paired off into two terminal nets based on adjacency of the terminals. The connectivity of a multiterminal net which spans more than one region is preserved by defining two terminal nets between adjacent regions.

In order to find the maximum set of independent nets in each region, we follow the approach used in selecting critical nets. That is, we create a permutation graph and find the maximum independent set in the permutation graph. Similar weight factors are also considered here to keep the routing regions as rectangular as possible.

The selection of these new nets increases the number of regions and decreases the set of nets to be routed. So far, we have considered routing in a topological sense, as no geometric positions have been assigned to any net. In the next step, we assign geometric positions to the set of nets, which define the regions, to obtain the geometric definition of the routing regions.

4.3.4 Terminal Assignments of Critical Nets

In this section we present an algorithm for the problem of terminal assignments of critical nets (TAC) with capability constraints.
Given a set of nets $L = \{n_1, n_2, \ldots, n_N\}$ which needs to be routed in a given channel, the set of terminals on the top terminal row is denoted by $TOP = \{t_1, t_2, \ldots, t_T\}$, and the set of terminals on the bottom terminal row is denoted by $BOT = \{b_1, b_2, \ldots, b_B\}$, respectively. Let us consider the boundary of the cells as an imaginary row of terminals denoted by $M = \{m_1, m_2, \ldots, m_P\}$, where $P$ is the number of terminal positions. Let $S = \{s_1, s_2, \ldots, s_K\}$ be the maximum independent set of critical nets selected in the previous two steps, where $K$ is the size of the independent set. The entire M2 routing area is partitioned into $K + 1$ regions, $R = \{r_1, r_2, \ldots, r_{K+1}\}$ by $S$, such that region $r_i$ is defined by nets $s_i$ and $s_{i+1}$. The region $r_1$ is defined by the left end of the channel and net $s_1$, similarly, $r_{K+1}$ is defined by $s_K$ and right end of the channel. According to the region partitioning, the set of top terminals $TOP$ is partitioned into $K + 1$ subsets $TOP = \{TOP_1, TOP_2, \ldots, TOP_{K+1}\}$, where $TOP_i = \{t_{i,1}, t_{i,2}, \ldots, t_{i,T_i}\}$. The set of bottom terminals $BOT$ is partitioned into $K + 1$ subsets $BOT = \{BOT_1, BOT_2, \ldots, BOT_{K+1}\}$, where $BOT_i = \{b_{i,1}, b_{i,2}, \ldots, b_{i,B_i}\}$. And the imaginary row of terminals $M$ is partitioned into $K + 1$ subsets $M = \{M_1, M_2, \ldots, M_{K+1}\}$, where $M_i = \{m_{i,1}, m_{i,2}, \ldots, m_{i,P_i}\}$, respectively. There are two types of capability constraints imposed on the assignments of terminals, namely the terminal capability constraints, which is modeled by a set $C_R$, and the M2 OTC river routing capability constraints, which is modeled by a set $C_{RS}$, respectively. And $LEN(S)$ is the objective function. Formally, an instance of the TAC problem is a 9-tuple

$$\Phi = (L, M, TOP, BOT, S, R, C_R, C_{RS}, LEN(S))$$

where

1. $L$ is the set of nets to be routed in the given channel;
2. $M = \{m_1, m_2, \ldots, m_P\}$, an imaginary row of terminals,
3. $P$ is the number of terminal positions;
4. \( TOP = \{t_1, t_2, ..., t_T, T \leq P\} \), the set of terminals on the top terminal row;

5. \( BOT = \{b_1, b_2, ..., b_B, B \leq P\} \), the set of terminals on the bottom of the channel;

6. \( S = \{s_1, s_2, ..., s_K\} \), the maximum independent set of critical nets selected in the previous two steps, where \( K \) is the size of the independent set.

7. \( R = \{r_1, r_2, ..., r_{K+1}\} \), the partitioned regions;

8. \( CR = \{c_{r_1}, c_{r_2}, ..., c_{r_{K+1}}\} \), the set of region terminal capability constraints.

9. \( CR_S = \{cr_{s_1}, cr_{s_2}, ..., cr_{s_{K+1}}\} \), the M2 OTC river routing capability constraints.

10. \( LEN(S) \) is the objective function.

A solution to \( \Phi \) is a function \( \pi = (f) \), where \( f : S \rightarrow \{1, 2, ..., P\} \), such that

1. \( f(c_{r_i}) \) satisfies \( CR \);
2. \( f(cr_{r_i}) \) satisfies \( CR_S \);
3. \( LEN(S) \) is minimum.
4. \( f \) is an injective function.

That is, a solution \( \Phi \) is an assignments of terminals in \( S \) to the imaginary row of terminals \( M \), satisfying the two capability constraints, while the objective function is minimum. Intuitively, \( f(s_i) = m_j \) means that both top and bottom terminals of \( s_i \) are assigned to the \( j \)th position (from the left) in the terminal imaginary row.

We first discuss the terminal capability constraint \( CR \). Each region \( r_i \) must has less or equal terminal number than the terminal positions between \( s_i \) and \( s_{i+1} \):

**Lemma 4** In a TAC Problem, the number of terminals \( T_i + B_i \) in a given region \( r_i \), must satisfy \( T_i + B_i \leq P_i \), where \( P_i \) is the number of terminal positions in \( r_i \), \( T_i \) and \( B_i \) are the number of top and bottom terminals within \( r_i \).
Proof: As each terminal in $TOP_i \cup BOT_i$ needs to be assigned to different terminal position, and terminals only can be assigned within its own region. Therefore, $T_i + B_i \leq P_i$. □

Based on this Lemma, we formalize the constraint $C_R$. First, we define the region capability $\text{cap}(r_i)$ as follows:

$$\text{cap}(r_i) = |m_{s_i} - m_{s_{i-1}}| - T_i - B_i$$

where

$T_i$ is the number of top terminals in region $r_i$;

$B_i$ is the number of bottom terminals in region $r_i$;

$m_{s_i}$ is the assigned terminal position of net $s_i$;

$m_{s_{i-1}}$ is the assigned terminal position of net $s_{i-1}$.

Using the region capability, a region terminal capability constraint can be defined as follows:

$$\text{cr}_i = \begin{cases} 1; & \text{if } \text{cap}(r_i) \geq 0 : \text{meaning } \text{cr}_i \text{ is satisfied.} \\ 0; & \text{if } \text{cap}(r_i) < 0 : \text{meaning } \text{cr}_i \text{ is not satisfied.} \end{cases}$$

Another capability constraint is the M2 OTC river routing constraint $CR_{S_r}$, which guarantees the routability of $S$ in M2 OTC area. In this constraint, we are given a fixed height routing region and a terminal row, each terminal is assigned a legal terminal assignment range on $M$, in order to guarantee the routability. For each net $s_i$, its top terminal $t_{s_i}$ is assigned a legal range $\text{range}_t(s_i)$ on $M$, and its bottom terminal $b_{s_i}$ is assigned a legal range $\text{range}_b(s_i)$ on $M$, respectively. A river routing constraint:

$$\text{cr}_{s_i} = \begin{cases} 1; & \text{if } m_{s_i} \in \text{range}_t(s_i) \cap \text{range}_b(s_i) : \text{meaning } \text{cr}_{s_i} \text{ is satisfied.} \\ 0; & \text{if } m_{s_i} \notin \text{range}_t(s_i) \cap \text{range}_b(s_i) : \text{meaning } \text{cr}_{s_i} \text{ is not satisfied.} \end{cases}$$

Use the algorithms presented in [9, 31], we can obtain the ranges of $S$ in $O(K)$ time, where $K$ is the size of $S$.

To satisfy both constraints at same time, we have the following Lemma:
Lemma 5 Given any two nets $s_i, s_j$ (assume $i < j$) and their terminal ranges $\text{range}_t(s_i), \text{range}_b(s_i), \text{range}_t(s_j), \text{range}_b(s_j)$, if $s_i$ is assigned to the range $\min(\text{range}_t(s_i), \text{range}_b(s_i))$ and $s_j$ is assigned to $\max(\text{range}_t(s_j), \text{range}_b(s_j))$, then a TAC problem is not feasible if

$$\sum_{k=i+1}^{j-1} \text{cap}(r_k) < 0$$

Where $\max()$ and $\min()$ represent the maximum and minimum value of two numbers.

Proof: As $s_i$ is assigned to its left-most position $\min(\text{range}_t(s_i), \text{range}_b(s_i))$, and $s_j$ is assigned to its right-most position $\max(\text{range}_t(s_j), \text{range}_b(s_j))$. Due to the river routing routability constraints, $s_i$ cannot move left and $s_j$ cannot move right. All the nets in $[s_i, s_j]$ must be assigned within the range:

$$[\min(\text{range}_t(s_i), \text{range}_b(s_i)), \max(\text{range}_t(s_j), \text{range}_b(s_j))]$$

However, $\sum_{k=i+1}^{j-1} \text{cap}(r_k) < 0$ means the $C_R$ constraint cannot be satisfied within this range, Therefore, the TAC problem is not feasible. $\square$

Based on Lemma 5, we use following algorithm to calculate the legal region $\text{range}(s_i)$ of $s_i$, such that $\text{range}(s_i)$ satisfies all the constraints. In the algorithm, $\text{range}L(s_i)$ denotes the left point of $\text{range}(s_i)$, while $\text{range}R(s_i)$ denotes the right point of $\text{range}(s_i)$, respectively.

Algorithm: $\text{Range}(st, end)$

$$st : \text{Start region number.}$$
$$end : \text{End region number.}$$

BEGIN

/* Calculate left point of each range. */
FOR $i = st$ TO $end$ DO
IF \( \text{rangeL}(s_{i-1}) + \text{cap}(r_i) < \min(\text{range}_t(s_i), \text{range}_b(s_i)) \)
THEN \( \text{rangeL}(s_i) = \min(\text{range}_t(s_i), \text{range}_b(s_i)) \)
ELSE IF \( \text{rangeL}(s_{i-1}) + \text{cap}(r_i) \in \text{range}_t(s_i) \cap \text{range}_b(s_i) \)
THEN \( \text{rangeL}(s_i) = \text{rangeL}(s_{i-1}) + \text{cap}(r_i) \)
ELSE IF \( \text{rangeL}(s_{i-1}) + \text{cap}(r_i) > \max(\text{range}_t(s_i), \text{range}_b(s_i)) \)
THEN \( \text{rangeL}(s_i) = -1 \)

/* Calculate right point of each range. */
\( \text{rangeR}(s_{K+1}) = 0 \)
FOR \( i = \text{end} \) TO \( \text{st} \) DO
IF \( \text{rangeR}(s_{i-1}) - \text{cap}(r_i) > \max(\text{range}_t(s_i), \text{range}_b(s_i)) \)
THEN \( \text{rangeR}(s_i) = \max(\text{range}_t(s_i), \text{range}_b(s_i)) \)
ELSE IF \( \text{rangeR}(s_{i-1}) - \text{cap}(r_i) \in \text{range}_t(s_i) \cap \text{range}_b(s_i) \)
THEN \( \text{rangeR}(s_i) = \text{rangeR}(s_{i-1}) + \text{cap}(r_i) \)
ELSE IF \( \text{rangeR}(s_{i-1}) - \text{cap}(r_i) > \min(\text{range}_t(s_i), \text{range}_b(s_i)) \)
THEN \( \text{rangeR}(s_i) = -1 \)
END

In case a net \( s_i \) does not satisfy Lemma 5, we simply remove it from \( S \).

Finally, the objective of TAC problem is to minimize a cost function \( L E N(S) \), which is defined as follows:

\[
L E N(S) = \sum_{i=1}^{K+1} |t_{s_i} - m_{s_i}| + |b_{s_i} - m_{s_i}| - |t_{s_i} - b_{s_i}|
\]

where

\( t_{s_i} \) is the top terminal position of \( s_i \);
\( b_{s_i} \) is the bottom terminal position of \( s_i \);
\( m_{s_i} \) is the assigned terminal position of net \( s_i \);
The function $\text{length}(i, j)$ represents the cost of assigning the net $s_i$ to the terminal position $m_j$, which is defined as follows:

$$\text{length}(r_i) = |t_{s_i} - m_j| + |b_{s_i} - m_j| - |t_{s_i} - b_{s_i}|$$

For a instance of $\Phi$, a solution $(i, j)$ (denoted by $(i, j)$—solution) represents the first $i$ nets in $S$, $\{s_1, s_2, \ldots, s_i\}$ are assigned to the first $j$ terminal position in $M$, $\{m_1, m_2, \ldots, m_j\}$, and the $\text{len}(i, j)$ denotes the minimum cost the solution. For a solution which does not satisfied the constraints, $\text{len}(i, j) = -1$.

We are now ready to present our algorithm for TAC problem. The input of the algorithm is an instance $\Phi = (L, M, \text{TOP}, \text{BOT}, S, R, C_L, C_R, C_{RS}, \text{LEN}(S))$ of the TAC problem, and the output is an optimal solution to $\Phi$, or an indication that no such solution exists.

Algorithm: $AssignC$

\[
\begin{align*}
\text{BEGIN} & \\
& \text{/* Initialization */} \\
& \text{FOR } i = 1 \text{ TO } K \text{ DO} \\
& \quad \text{len}(i, 0) = -1; \\
& \text{FOR } j = 0 \text{ TO } P \text{ DO} \\
& \quad \text{len}(0, k) = 0; \\
& \text{/* Computing objective function len}(K, P) \text{ using dynamic programming */} \\
& \text{FOR } j = 1 \text{ TO } P \text{ DO} \\
& \quad \text{FOR } i = 0 \text{ TO } K \text{ DO} \\
& \quad \quad \text{/* Based on the previous net assignments, */} \\
& \quad \quad \text{/* Compute legal ranges for the assigning net. */} \\
& \quad \quad \text{Range}(i, K) \\
& \quad \quad \text{IF assignment is in legal range} \\
& \quad \quad \quad \text{THEN } \text{len}(i, j) = \min\{\text{len}(i, j - 1), \text{length}(i, j) + \text{len}(i - 1, j - 1)\}
\end{align*}
\]
ELSE \( \text{len}(i,j) = -1 \)

IF \( \text{len}(K,P) = -1 \)
THEN RETURN "\( \Phi \) is not feasible".
ELSE RETURN \( \pi = \text{len}(K,P) \)
END

**Theorem 8** Algorithm AssignC solves the TAC problem in \( O(PK) \) time and \( O(PK) \) spaces, where \( P \) is the number of terminal positions and \( K \) is the size of the selected independent set.

**Proof:** We first prove the correctness of the algorithm. We prove by induction on \( j \) that if \( \Phi \) has a solution \((i,j)\), then \( \text{len}(i,j) \) is equal to the minimum all the \((i,j)\) solution to \( \Phi \) so far.

This is true for \( j = 0 \) where we chose the boundary values. Assume it is true for \( j - 1, 1 \leq j \leq K \). If \( \Phi \) has a \((i,j)\)-solution with minimum cost function, then it must be either an \((i,j-1)\)-solution or the cost of \((i-1,k-1)\)-solution plus \( \text{length}(i,j) \). By the induction hypothesis and the fact \( \text{len}(i,j) = \min\{\text{len}(i,j-1), \text{length}(i,j) + \text{len}(i-1,j-1)\} \), \( \text{len}(i,j) \) still holds the minimum cost \((i,k)\)-solution of \( \Phi \).

Therefore, the algorithm AssignC solves the TAC problem \( \Phi \).

The time complexity of the algorithm is clearly dominated by the computation of the function \( \text{len}(i,j) \). As the computation of \( \text{len}(i,j) \) can be finished in \( O(PK) \) time and \( O(PK) \) spaces, where \( P \) is the number of terminal positions and \( K \) is the size of the selected independent set. □

### 4.3.5 Terminal Assignments of Non-Critical Nets

In this section, we describe the terminal permutation algorithm used by our router.
Given two sets $T$ and $B$, where $T = \{T_1, T_2, \ldots, T_{k+1}\}$ and $B = \{B_1, B_2, \ldots, B_{k+1}\}$. Each $T_i(B_i)$ consists of a non-permutable list of terminals. We need to find a permutation of $P = \{P_1, P_2, \ldots, P_{k+1}\}$, such that each $P_i$ is a shuffle of $T_i$ and $B_i$. The objective function of the shuffle operation is to minimize horizontal constraints and maximize the bipartite subgraph in the overlap graph.

We adopt the following heuristic. We start with the permutation $P_i = T_i B_i$, for all $i$. Then shuffle each pair $T_i B_i$. Inductively, assume of $T_1 \ldots T_{i-1}$ and $B_1 \ldots B_{i-1}$ have been pairwise shuffled. As for the basis, the first shuffle $P - 1$ is $T_1 B_1$. We obtain $P_i$ as follows. Consider $T_i = \{t_1, \ldots, t_s\}$ and $B_i = \{b_1, \ldots, b_s\}$. There are $s + 1$ positions to which $t_1 \ldots t_s$ can be assigned to assuming $b_1 \ldots b_{s-1}$ are assigned to the left of $t_s$ and $b_{s+1} \ldots b_s$ are assigned to the right of $t_s$. For each position, we find the number of independent nets created.

Then we form a matching diagram [13]. On one side we have the set of intervals $[-\infty, t_1], [t_1, t_2], \ldots, [t_s, +\infty]$ and on the other side we have $b_1, \ldots, b_s$. Each edge has the weight calculated before, that is, the number of independent nets corresponding to that position. Now, we find a maximum-weight non-cross matching in $O(s\log s)$ time [13]. The matching dictates a shuffle. This finishes the inductive step, that is, finding a shuffle of $T_i$ and $B_i$.

Once the terminals have been permuted, we need to select a subset of nets, for routing over the cells. Since only M3 is available, only planar routing is allowed. However, we need to select two planar sets, one for routing over the top row of cells and one for routing over bottom row of cells. In order to find such planar sets, we define an overlap graph $G = (V, E)$, as follows:

$$V = \{v_i | \text{vertex } v_i \text{ corresponds to interval } I_i \text{ representing } n_i\}$$

and

$$E = \{(v_i, v_j) | \text{if intervals representing nets } n_i \text{ and } n_j \text{ overlap}\}$$

Where, $I_i$ is an interval defined on a real line by using the two terminals of the net $n_i$, as end points. The overlap graph for the channel defined in Figure 4.3
and Figure 4.4 is shown in Figure 4.5. It can be easily seen that a set of nets is planar if and only if it forms an independent set in $G_O$. Similarly, a MBS in $G_O$ corresponds to a set of nets routable in two layers in a planar fashion. However, the problem of finding a MBS in an overlap graph is known to be NP-Hard [17]. Therefore, we use an approximation algorithm developed in [17], which produces a MBS of an overlap which is $0.75 \times OPT$, where $OPT$ refers to the size of the optimal solution.

4.3.6 River Routing of M2 OTC Area

After terminal assignments is done, the routing problem in M2 over the cell area is a standard river routing problem [9, 31].

Given a set of nets $L$, and fixed number of routing tracks decided by the cell design — in this chapter, we assume that there are 12 tracks available on both sides — using the algorithms presented in [9, 31], we can check the routability and finish the river routing with minimum jogs in $O(n)$ time, where $n$ is the number of nets. (We omit details of this phase and refer the reader to the cited references above, for details).

4.3.7 Net Selection for M3 Layer

In this section, we explain the procedure of M3 net selection. The objective of this operation is to minimize the channel height. Because we already eliminated all vertical constraints, the channel height is decided by the size of maximum clique in the horizontal constraint graph ($H_{max}$).

As we don't allow vias in OTC area, we only can route two planar subsets of nets in M3 OTC area. We need select this two sets of nets, such that $H_{max}$ of remaining nets is minimized. Basically, this problem is a maximum two independent sets (2MIS) problem in a circle graph, with set size constraint, and objective function is to minimize the $H_{max}$ of remaining nets. As finding optimal
Figure 4.4 Terminal Assignments of Critical Nets.
Figure 4.5 Example of Overlap Graph.
2MIS in a circle graph has already be shown to be NP-complete, and 0.75 * OPT approximation algorithm has been presented in [17], we use a approach similar to one reported in [17].

In this chapter, we find the maximum independent set (MIS) one by one. We run this algorithm two times to obtain two MIS's which is at least 50% of optimal 2MIS.

Given a set of two terminal nets on a single row, \( L \) which contains \( N \) nets, the terminals are numbered from 0 to \( 2N - 1 \) from left to right on the single row. We use \( n_{i,j} \) to denote the net having terminals numbered \( i \) and \( j \). Let

\[
V = \{v_{i,j} : i < j \text{ and } i, j \in L\} \quad \text{and} \quad E = \{(v_{i_1,j_1}, v_{i_2,j_2}) \in V^2 : i_1 < i_2 < j_1 < j_2 \text{ or } i_2 < i_1 < j_2 < j_1\}
\]

In other words, each vertex \( v_{i,j} \in V \) corresponds to a net \( n_{i,j} \in L \), and there is an edge in \( E \) for each pair of vertices whose corresponding nets intersect. Therefore, the graph \( G(V, E) \) is a circle graph associated with \( L \). If \( 0 \leq i, j \leq 2N - 1 \), then we use \( G_{i,j} \) to denote the subgraph of \( G \) induced by the set of vertices \( \{v_{i,m} \in V : i \leq l, j \leq m\} \).

Let \( MIS(i,j) \) denote an independent set of \( G_{i,j} \), such that the \( H_{\text{max}} \) of nets in \( MIS(i,j) \) satisfies the size constraint, and the \( H_{\text{max}} \) of the remaining nets \( L - MIS(i,j) \) is minimum. Note that if \( j \leq i \), then \( G_{i,j} \) is the empty graph and, hence, \( MIS(i,j) = \emptyset \).

The algorithm is an application of dynamic programming. In particular, \( MIS(i,j) \) is computed for each pair \( i, j \); \( MIS(i,j_1) \) is computed before \( MIS(i,j_2) \) if \( j_1 < j_2 \). To compute \( MIS(i,j) \), we let \( k \) be the unique number such that \( n_{k,j} \in L \) or \( n_{j,k} \in L \). If \( k \) is not in the range \([i, j - 1]\), then \( G_{i,j} = G_{i,j-1} \), and, hence, we set \( MIS(i,j) = MIS(i,j - 1) \). If \( k \) is in the range \([i, j - 1]\), then:

\[
MIS(i,j) = MIS(i,k - 1) \cup \{v_{k,j}\} \cup MIS(k + 1, j - 1).
\]

For this new \( MIS(i,j) \), we need to calculate the \( H_{\text{max}} \) of it and check the size constraint. If its \( H_{\text{max}} \) does not satisfy the size constraint, we drop this \( MIS(i,j) \)
by setting $MIS(i, j) = MIS(i, j - 1)$. Otherwise, we remove $MIS(i, j)$ from $L$, recalculate the $H_{\text{max}}$ of remaining nets. If this the $H_{\text{max}}$ of remaining nets is bigger than the previous one, we still need to drop this $MIS(i, j)$ by setting $MIS(i, j) = MIS(i, j - 1)$.

More formally, the algorithm is as follows:

Algorithm $HMIS$

BEGIN

FOR $j = 0$ TO $2N - 1$ DO

BEGIN

$MIS = \emptyset$

/* Compute $MIS(i, j)$ for each $i < j$ */

$k =$the number such that $k, j \in L$ or $j, k \in L$;

FOR $i = 0$ TO $j - 1$ DO

BEGIN

IF $i \leq k \leq j - 1$

THEN $MIS(i, j) = MIS(i, k - 1) \cup \{v_{k, j}\} \cup MIS(k + 1, j - 1)$

IF $h_{\text{max}}(MIS(i, j)) < \text{SIZE}$

and $h_{\text{max}}(L - MIS(i, j)) < h_{\text{max}}(L - MIS)$

THEN $MIS = MIS(i, j)$

ELSE $MIS(i, j) = MIS(i, j - 1)$

END

END

$H_{\text{max}}(l)$

BEGIN

$H = 0, H_{\text{max}} = 0$

FOR $i = 0$ TO $2N - 1$ and $i \in l$ DO

BEGIN
IF \( i \) is the LEFT TERMINAL of a net
THEN \( H = H + 1 \)
IF \( H > H_{\text{max}} \) THEN \( H_{\text{max}} = H \)
ELSE \( H = H - 1 \)
END
\( \text{return}(H_{\text{max}}) \)
END
END

Finally, since \( G = G_{0,2N-1} \), we have that \( MIS \) is a maximum independent set of \( G \).

4.3.8 The Performance Bounds of ICR-3

In this section, we prove the performance bounds of ICR-3 in terms of channel height and number of vias.

**Theorem 9** In the layout created by ICR-3, the number of vias is \( T + B - 2K \), where \( T \) is the total number of top terminals, while \( B \) is the total number of bottom terminals, and \( K \) is the size of the maximum independent set of critical nets for \( M_2 \).

**Proof:** According to algorithm IRC-3, the critical nets for \( M_2 \) are routed straight in the channel on \( M_2 \), therefore, no vias are needed for these nets. All the remaining nets need one via per terminal to complete the over-the-cell routing or HVH channel routing. Therefore, the number of vias is \( T + B - 2K \), as all critical nets are two terminal nets. □

**Theorem 10** In the layout created by ICR-3, the number of tracks needed in the channel is \( H_{\text{max}}/2 \), where \( H_{\text{max}} \) is the size of the maximum clique in the horizontal
constraint graph defined by the nets in set \( L - MIS_2 - MIS_3 \), \( L \) is a given set of nets, \( MIS_2 \) is the maximum independent set of critical nets for \( M_2 \), and \( MIS_3 \) is the two maximum independent set of nets for \( M_3 \).

**Proof:** In ICR-3, nets in \( MIS_2 \) are routed straight in the channel and make no contribution to the channel height. While nets in \( MIS_3 \) are routed in OTC area on \( M_3 \), also add nothing to the channel height. All the remaining nets \((L - MIS_2 - MIS_3)\) are routed in the channel and as we don’t have any vertical constraints, using *left-edge* algorithm, we can route all the remaining nets in \( H_{max}/2 \) tracks using HVH routing model. □

4.4 Formal Statement of the Algorithm

In this section, we present the formal details of our algorithm. For sake of brevity, we provide details of only those phases which are not discussed earlier. It must be pointed out that once we have determined the set of nets to be routed over the cells it is rather easy to assign tracks to them so that they can be routed with violating any design rules. The sets of nets which are not routed in any of the previous phases is routed in the channel, using a HVH routing model. Since there are no vertical constraints, we simply use a *left-edge* algorithm, which is modified to assign nets to both \( M_1 \) and \( M_3 \) in a left to right scan. Complete layout of example in Figure 4.3 is shown in Figure 4.2.

**Algorithm ICR-3()**

Input: \( N = \{n_1, n_2, \ldots, n_m\} \) is a set of nets.

Output: Over-the-cell channel routing.

*begin Algorithm*

**PHASE 1: Net Classification.**
PHASE 2: Net Weighting and Finding the maximum independent set in Permutation Graph.
Construct_Permutation_Graph(G, N)
Weights_Assignment_Graph(G, N)
\( N_1 = \text{Max Independ. Set}(G) \)
\( N_2 = N - N_1 \);

PHASE 3: TYPE III nets Partitioning.
/* explained in Section 5.3 */

PHASE 4: Terminal Assignments of Critical Nets and Region Partitioning.
Details in Section 5.3 *

PHASE 5: Terminal Assignments of Non-Critical Nets.
See Section 5.5 for details of this step *

PHASE 6: M2 River Routing.
/* explained in Section 5.6 */

PHASE 7: Finding the MBS in a circle graph.
Construct_Circle_Graph \((G_c, N_2)\)
\( N_3 = \text{HMIS}(G_c) \)
\( N_4 = \text{HMIS}(G_c - G_{N_3}) \)
\( N_5 = N_2 - N_3 - N_4 \);

PHASE 8: Over-The-Cell Routing.
This phase is similar to other OTC routers [6, 17]

PHASE 9: Three Layer Channel Routing

Let us sort the nets in $N_5$. Let $n_1, n_2, \ldots, n_m$ be a list of nets sorted on their left terminals. Let $t^l_j$ refer to $j^{th}$ track on $l^{th}$ layer. The procedure INTERSECT($n_i, t^l_j$) returns true if net $n_i$ intersects another net already routed on track $t^l_j$.

For each $n_j \in L_5$ do

j = 1; assigned = FALSE

While ($j \leq \text{max\_track}$)

If (INTERSECT($n_i, t^l_j$) $\neq$ TRUE) Then

ASSIGN($n_i, t^l_j$); assigned = TRUE;

Else If INTERSECT($n_i, t^l_3$) $\neq$ TRUE Then

ASSIGN($n_i, t^l_3$); assigned = TRUE;

Else $j = j + 1$;

If (assigned = FALSE) Then

max\_track = max\_track + 1; $j = \text{max\_track}$

ASSIGN($n_i, t^l_j$); assigned = TRUE;

PHASE 10: Clean up Routing

end Algorithm

The worst-case time complexity of algorithm ICR-3 is determined by algorithm HMIS which has time complexity $O(n^3)$ where $n$ is the number of nets. In practice, however, algorithm runs very fast as indicated in Section 9.
4.5 Experimental Results

Algorithm ICR-3 has been implemented on a SPARC 1+ workstation in the C programming language. It was tested extensively on benchmark examples including PRIMARY I and PRIMARY II. For an example, we show the performance of ICR-3 on the PRIMARY I benchmark in comparison with a conventional two layer router [25], a two layer over-the-cell router [6], and a conventional three layer channel router [7]. The placement of the PRIMARY I example was obtained from TimberWolfSC Version 5.1 [19], and the global routing is from [4].

Algorithm ICR-3 on the average, produces a 73% reduction in channel height as compared to a conventional two layer channel (2CRP) router, a 58% reduction as compared to a two layer over-the-cell (2OTC) router, and a 47% reduction as compared to a conventional three layer channel (3CRP) router. For PRIMARY I, it produces a routing which are 65%, 51% and 40% better than 2CRP, 2OTC and 3CRP routers, respectively. The percentage improvement of ICR-3 over 2CRP, 2OTC, and 3CRP for the channels of PRIMARY I is listed in Table 4.1. An example routing of channel 14 of PRIMARY I is shown in Figure 4.6.

Algorithm ICR-3 is also very successful in reducing the number of vias in a layout. Via minimization is an inherent feature of the ICR-3 routing model due to the planarity requirement for nets in the over-the-cell routing region. Our over-the-cell routing algorithm typically reduces the number vias per routing by 56% as compared to 3-CRP router. For the PRIMARY I benchmark, ICR-3 produces a solution, which 50%, 28% and 56% better than 2CRP, 2OTC and 3CRP. Via minimization details for each channel of the PRIMARY I example are shown in Table 4.2.

Algorithm ICR-3 not only produces excellent routings, it is also very efficient. For the entire PRIMARY I chip, ICR-3 only needs 5.8 seconds running time. The highest density channel (channel no. 8) is routed in 0.4 seconds. This
Table 4.1

Experimental Results for PRIMARY 1: Channel Height

<table>
<thead>
<tr>
<th>Chan. No.</th>
<th>% of Vac.</th>
<th>No. of Tracks Produced</th>
<th>% Impro. by ICR-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2CRP</td>
<td>2OTC</td>
</tr>
<tr>
<td>1</td>
<td>85</td>
<td>11</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>74</td>
<td>16</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>24</td>
<td>21</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>21</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>52</td>
<td>23</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>58</td>
<td>22</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>24</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>53</td>
<td>21</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>63</td>
<td>17</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>64</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>62</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>14</td>
<td>64</td>
<td>13</td>
<td>9</td>
</tr>
<tr>
<td>15</td>
<td>63</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>67</td>
<td>11</td>
<td>7</td>
</tr>
<tr>
<td>17</td>
<td>66</td>
<td>14</td>
<td>8</td>
</tr>
<tr>
<td>18</td>
<td>91</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td>298</td>
<td>214</td>
<td>172</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Channel 14 of PRIMARY 1

Number of Tracks: 4; Number of Vias: 201.

Figure 4.6 Complete Routing Created By ICR-3.
Table 4.2

Experimental Results for PRIMARY 1: Via Minimization

<table>
<thead>
<tr>
<th>Channel No.</th>
<th>% of Vacant Terminals</th>
<th>No. of Vias</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2CRP</td>
</tr>
<tr>
<td>1</td>
<td>85</td>
<td>150</td>
</tr>
<tr>
<td>2</td>
<td>74</td>
<td>339</td>
</tr>
<tr>
<td>3</td>
<td>64</td>
<td>486</td>
</tr>
<tr>
<td>4</td>
<td>60</td>
<td>534</td>
</tr>
<tr>
<td>5</td>
<td>64</td>
<td>539</td>
</tr>
<tr>
<td>6</td>
<td>52</td>
<td>662</td>
</tr>
<tr>
<td>7</td>
<td>58</td>
<td>612</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>714</td>
</tr>
<tr>
<td>9</td>
<td>53</td>
<td>608</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
<td>526</td>
</tr>
<tr>
<td>11</td>
<td>63</td>
<td>468</td>
</tr>
<tr>
<td>12</td>
<td>64</td>
<td>462</td>
</tr>
<tr>
<td>13</td>
<td>62</td>
<td>511</td>
</tr>
<tr>
<td>14</td>
<td>64</td>
<td>440</td>
</tr>
<tr>
<td>15</td>
<td>63</td>
<td>439</td>
</tr>
<tr>
<td>16</td>
<td>67</td>
<td>377</td>
</tr>
<tr>
<td>17</td>
<td>66</td>
<td>429</td>
</tr>
<tr>
<td>18</td>
<td>91</td>
<td>81</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>8146</td>
</tr>
</tbody>
</table>

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
fast speed along with its superior performance makes ICR-3 router a practical router for real world problems.

4.6 Summary

In this chapter, we have developed a three layer over-the-cell router. The key feature of this router is use of a new cell model. Our router, ICR-3, outperforms all existing routers not only in terms of channel height but also in terms of vias. In addition, the router is very efficient and can route “real-world” designs in a few seconds.
CHAPTER V

CONCLUSIONS AND FUTURE RESEARCH

5.1 Conclusions

In this thesis, we basically have two results. First, we develop several performance-bounded algorithms for the classical single row routing problem. The approach is based on a graph theoretic representation, in which an instance of the single row routing problem is represented by three graphs: an overlap graph, a containment graph, and an interval graph. Second, we develop a new three layer over-the-cell router based on new cell model. This is the first channel router based on this cell model. Our router out performs all existing channel routers, and can actually complete chip layout in seconds.

5.2 Future Research

A number of problems related to single row routing and over-the-cell routing remain open.

First, most of the previous research has been directed towards solving single routing problems with one objective function. It would be very interesting if we can develop some algorithms to consider several objective functions at same time.

Another area open for future research is the development of new decomposition schemes following our approach. In this context, decomposition based on cut-vertex in the overlap graph and clique-separators in the interval graph can be studied.

The development of over-the-cell routers based on new cell model provides us with new research directions. Industry need a provable channelless router based on the new cell model. Some integrated approaches to global and channel routing are also of great interests.
REFERENCES


83

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.


