OPTIMIZED

UNIDIRECTIONAL ROUTING

H. W. CARTER

DIGITAL INTEGRATED SYSTEMS CENTER REPORT
DISC/81-2

DEPARTMENT OF ELECTRICAL ENGINEERING - SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

September 1981

This work was supported in part by the National Science Foundation under Grants ENG 74-18647 and ECS-8005957, and the DOD under a VHSIC Phase 3 Contract No. DAAK20-80-C-0278 administered by the Department of the Army, Fort Monmouth, New Jersey.

OPTIMIZED
UNIDIRECTIONAL ROUTING

H. W. CARTER

DIGITAL INTEGRATED SYSTEMS CENTER REPORT
DISC/81-2

DEPARTMENT OF ELECTRICAL ENGINEERING - SYSTEMS
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIFORNIA 90007

September 1981

This work was supported in part by the National Science Foundation under Grants ENG 74-18647 and ECS-8005957, and the DOD under a VHSIC Phase 3 Contract No. DAAK20-80-C-0278 administered by the Department of the Army, Fort Monmouth, New Jersey.

OPTIMIZED UNIDIRECTIONAL ROUTING

by

Harold Wesley Carter

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the

Requirements for the Degree

DOCTOR OF PHILOSOPHY

(Electrical Engineering)

January 1980

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90007

This dissertation, written by

HAROLD WESLEY CARTER

under the direction of his Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements of the degree of

DOCTOR OF PHILOSOPHY

L. G. Oxnard

Dean

Date 1-9-80

DISSERTATION COMMITTEE

Melvin D. Braun
Chairman

John P. Hayes

Dennis R. Estes

FOREWARD TO DISC REPORT NO. 81-2

This report presents a new routing algorithm for electric output carriers, such as printed circuit boards and LSI chips. Previous routing techniques typically fall into one of three categories, namely maze, line or channel routers. In this work a new representation of the routing problem is found — namely a graph model. The results derived from this model are extremely powerful, since they allow for the following operations to be easily performed:

1. minimization of length, wire congestion or perturbation (re-assignment of wires to new tracks).
2. routing with or without fixed track capability.
3. re-assignment of wires to new tracks.

The major new concept developed and employed is an access graph. Though the complexity of the routing technique presented here is similar to that of the Lee algorithm, its ability to re-route wires significantly enhances its capability.

Melvin A. Breuer
Professor
University of Southern California
19 June 1981

ABSTRACT

This dissertation presents and analyzes graphical methods for routing printed circuit board configurations which consist of points which lie on a straight line. Three types of routing configurations are investigated: unconstrained configurations which contain no tracks; fixed-track configurations in which existing routed wires are fixed permanently to the tracks in which they have been assigned; and floating configurations in which existing routed wires are reassigned to free tracks to accommodate the routing of new wires.

Five independent objectives in routing a new wire between two points in a configuration are explored. The objectives which apply to unconstrained configurations are minimum wire length, minimum number of switches in a wire path between the upper and lower areas, and wire routing such that no switches exist in the path for that wire and the number of wires in the upper and lower areas is approximately equal. The objectives which apply to fixed-track configurations are minimum wire length and minimum wire congestion. The objectives considered for floating-track configurations are the same as for fixed-track configurations plus the routing of each wire such that the minimum

Department of the Army Superior Cadet Decoration Award

The Department of the Army provides this award annually to an outstanding ROTC cadet in each year of Military Science at an institution. The class advisors for each year group will present the awards.

Association of the United States Army (AUSA) Award

Presented by Colonel Michael Gray, representing the Greater Los Angeles Chapter of AUSA.

The AUSA Award

A medal pendant presented to the outstanding MS III cadet who contributed the most toward advancing the standing of the Military Science Department at USC.

AUSA Outstanding Cadet Awards

A plaque and $50.00 will be presented to an outstanding MS I, II, and III cadet ranking in the top third of their year groups' Order of Merit List.

Special Awards

Presented for special achievements in various aspects of ROTC activities.

* * * PROGRAM * * *

COCKTAILS

COLORS

INVOCATION

TOASTS

WELCOME

Four Man Drill Team Exhibition

DINNER

Ceasar Salad
New York Strip Steak
Baked Potato
Julienne Vegetables
Rolls & Butter
Raspberry Trifle

PRESENTATION OF AWARDS

Music by Eric Erickson

number of existing routed wires is moved (perturbed) to accommodate the new wire.

The major graphical method presented is to generate a graph representing the structure of the configuration under consideration, determine a path in the graph which represents the path in the configuration to be followed by a new wire, assign the wire to tracks, and update either the configuration or graph which contains the newly routed wire such that the next new wire may be routed. All routing objectives with the exception of the no-switch routing objective are obtained by using variations of Dijkstra's minimum-cost path algorithm to find the path in the graph. The no-switch routing objective is obtained by appropriately bicoloring the graph and adding a new node in the graph which represents the location of the new wire in the configuration.

Detailed procedures are given for implementing the methods described above for all configuration types and routing objectives. Theoretical analysis shows that the algorithmic complexity of most of the procedures is $O(n\log n)$ in both execution time and memory space. The results of two comprehensive experiments using a computer program implementing most of the procedures presented in the dissertation demonstrates that both the procedures and the theoretical complexity analysis are valid.

A unique application of the cellular approach to minimum-wire length routing in a configuration is also presented. Using differing-sized rectangular cells, a Lee-type routing algorithm is given which has similar theoretical performance as the graphical approach described above.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.0 Overview

Computers are often used in the layout of electronic assemblies. As printed circuit boards and integrated circuits have become increasingly complex, the use of design automation programs to partition networks, place components and route interconnections have become indispensible. Printed circuit board layout is a particularly complex problem and has absorbed the attention of researchers for the past several decades. Several comprehensive bibliographies and tutorials on the subject of design automation, which includes automated methods for routing, have been compiled by Breuer [1,2] and Hightower [8].

The layout of printed circuit (PC) boards is accomplished in several distinct steps. First, the components required to realize the given network design are allocated to one or more PC boards. If more than one PC board is needed, the network is partitioned, often using design automation programs [3], such that the components implementing each network partition are properly allocated to their respective PC boards. Constraints such as board size and

the number of edge-pins are usually major factors in the partitioning process.

Next, the components are assigned to specific locations on the board. It is desired to place the components such that a high completion rate results when routing the interconnections between the component pins on the board. Again, automated methods [4,5] exist for placing components on PC boards. Unfortunately, the correlation of placement with routing completion rate is not predictive in general.

The last phase of layout is to actually route the paths for the interconnections (etch) on the PC board. This function may be performed manually, automatically using a computer, or interactively.

The primary goal in routing is to maximize the number of nets routed at the least cost. A secondary goal is to perform the routing function completely by automated means (i.e., achieve 100% completion rate with no human intervention in the process). The algorithms implementing these automated methods are called routers, and they tend to be heuristic in nature. Completion rates in practice are high (>95%) although as a board becomes more densely packed with components completion rates decrease.

1.1 Automated Methods of Routing

1.1.1 The interconnection problem

The interconnection problem can be subdivided into four ordered subproblems [2]: wirelist determination, layering,

ordering, and wire layout. Wirelist determination, layering, and ordering will not be discussed here although their solution is necessary before wire layout can be performed. A survey of these problems and their known solutions is given in Akers [9].

1.1.2 Wire layout methods

There exist three basic classes of wire layout algorithms — all characterized by the method or model employed to find minimum-cost paths. These three classes are:

1. cellular
2. channel, and
3. line probe.

1.1.2.1 Cellular layout methods

The cellular approach is best illustrated by the well-known Lee algorithm [6] which is based upon some results by Moore [7]. The Lee algorithm lays out a grid of squares on the board surface, encodes each cell with an appropriate integer representing its minimum Manhattan distance from a given source cell, and then retraces from a terminal cell to the source cell using the cell coding to find a minimum-length path. Numerous modifications have been made to the Lee algorithms to reduce computation time [10], computer storage requirements [11], and to extend its application to such problems as multi-layer routing [12] and variable-weight minimum-cost paths [9].

Routing methods using graphs in the solution of wire layout problems are related to cellular methods. As an example, Van Cleemput [13] describes a graph-theoretic model and procedure for routing single-sided boards and integrated circuits. Kuh [14] suggests a graph-theoretical approach to the interconnection of points lying in a line.

1.1.2.2 Channel layout methods

The channel class of wire layout algorithms is best characterized by Foster [15] for the fixed via model and by Stevens [16] for the floating via model. Foster's model requires that the pins and vias on the board be in fixed locations and arranged in a matrix of rows and columns. In addition, path segments only connect two points in the same row or column. Vertical and horizontal segments are placed on separate planes. Finally, path segments may occupy either side, but not both sides (by switching between points) of the line of point to which the segment is attached.

The routing process is accomplished in three steps:

1. break multipoint nets into two-point nets,
2. find vias to break two-point connections into horizontal and vertical segments, and
3. place segments in a channel and assign them to tracks.

Foster claims a 100-fold improvement in CPU time over the

Lee algorithm.

A channel routing method similar to that of Foster is described by So [22] and Ting et al. [23]. The board model is constrained to have fixed points arranged in a matrix, the areas between rows and columns of points are called streets rather than channels, and segments may switch between the two streets adjacent to the point line containing the points to which the segments are attached. Heuristic algorithms are presented which determine the appropriate path for a two-point segment using a description of both the space available in the streets adjacent to each point and the number of segments attached to each point. The algorithm does not route optimally, and it is incomplete for certain configurations [14]. A new solution to the problem presented by Ting is alluded to in [14] but no efficient procedure yet exists to implement their ideas.

Stevens' approach is similar to Foster's except that vias are not constrained to be fixed. The approach is to assign segments to channels using heuristic procedures to resolve segment-end conflicts and locate the vias. The segments are then allocated to tracks in the channels such that the number of tracks used is minimal.

Channel routing has been of interest to researchers primarily because of its inherent speed and its ability to route efficiently. However, it does not guarantee minimum-cost paths as does the Lee algorithm. Several other

channel routing approaches may be found in Mah [17] and Lass [18].

A routing approach which uses a combination of both graphical and channel techniques is given by Hitchcock [21]. The algorithm divides the board into octagons such that pins are located at the corners. More than one wire may traverse an octagon. A Lee-type expansion is performed on the octagons to find paths. After all paths are assigned to octagons, track assignment within octagons is performed in a manner similar to the methods used in channel routers. This routing approach saves computer memory space over that of the Lee algorithm and has the benefit of providing better board utilization since the final position of each wire is not fixed until all paths within an octagon have been found.

1.1.2.3 Line-probe layout methods

The last of the three classes of layout methods is the line-probe method. A classical paper by Hightower [19] illustrates this method. The line-probe method essentially considers the board to be a continuum (rather than divided into cells or channels as for the graphical and channel methods). It constructs a sequence of line segments emanating from two points to be connected. When line segments from the two intersect a path has been found. A retrace procedure is then invoked to find the shortest path back

through the sequence of line segments. The line-probe algorithm is claimed to perform as well as the Lee algorithm with up to a 100-fold decrease in execution time [8]. Extensions to the basic Hightower algorithm have been made by Mattison [20] who maximizes utilization of the board by assigning different units to different boundary types, and by Mikami [9] who performs an exhaustive search for all paths using the line-probe algorithm thereby guaranteeing a path will be found if one exists.

1.2 The Unidirectional Routing Problem

A very interesting and practical routing problem is the unidirectional routing problem, so called because the basic routing configuration is a single line of points lying on a single board layer. Practical printed circuit boards can be characterized by an orderly arrangement of unidirectional point lines as illustrated in Figure 1.1. A point represents either a pin or a via. If we restrict a path between any two points in this configuration to consist of subpaths each of which connects two colinear points, then a possible method for routing this path is to (1) determine the set of colinear points which define the subpath end points, and (2) route each subpath. For example in Figure 1.1 a path between points A and B is shown which consists of three subpaths (A-a,a-b,b-B). One of the subpaths is routed such that it "switches" between the point line.

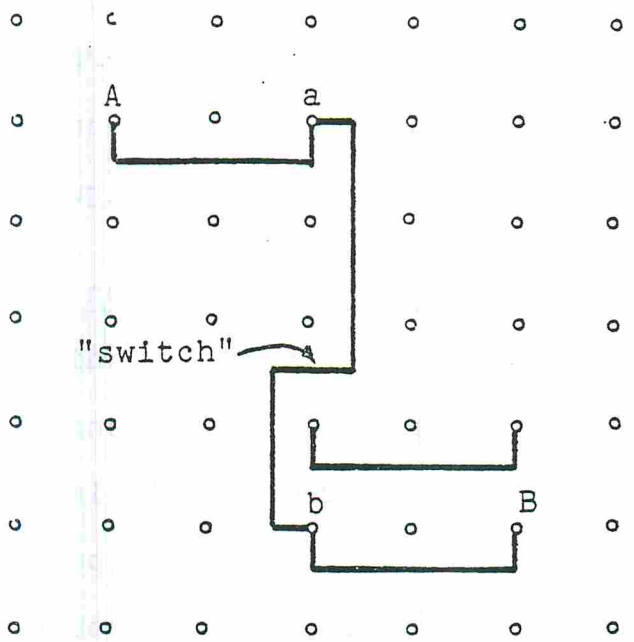


Figure 1.1. A single-layer printed circuit configuration.

The unidirectional configuration model is very general in that it can support the routing of multipoint nets. Multipoint nets are decomposed into the two-point nets which are then divided into subpath segments. Our primary interest in this dissertation is to optimally and efficiently route the subpath segments on unidirectional lines of points. The details of the general unidirectional routing problem and possible solutions are presented in So [22] and Kuh [14] and briefly discussed in Chapter 2 of this

work. In contrast to the rather heuristic solution approach adopted by these researchers, the approach taken here is based on finding minimum-cost paths in a graphical model of the unidirectional configuration.

1.3 Overview of this Dissertation

The techniques described in this dissertation optimally and efficiently implement solutions to the unidirectional routing problem. In general, it is assumed that the configuration consists of a line or points on a single layer with possibly some nets already routed.

Every technique given in this study assumes an ordered net list defined on the line of points is available. Each net in the list is described by a paired value (a,b) which represents the end points of that net. We call these nets *two-point nets*. Within this context, a multipoint net defined on more than two points in the point line is made up of two or more two-point nets. Given this ordered set of nets, each net is routed, one after another, such that a particular local objective function of each path for the net is met. The objective functions considered in this study are: minimum path length, minimum routing congestion, minimum routes perturbed (to free tracks for new routes), and minimum switching paths. Also, three different board constraints are considered as described below.

Chapter 2 presents the major definitions and notation used throughout the rest of this dissertation as well as previous results on this problem. In addition, a graphical model of the unidirectional configuration is introduced. Finally, a least-cost path algorithm is presented.

Chapters 3, 4 and 5 deal with the unidirectional routing problem under three different sets of constraints, and give procedures for routing in each case. In Chapter 3, the unconstrained case is considered where tracks are ignored and the areas which may contain routes are considered unbounded. Paths are routed such that their lengths are minimized. Chapter 4 deals with the case, called fixed-track unidirectional routing, where the areas which may contain routes are limited in size, and paths once assigned to tracks cannot be moved. Paths are routed such that either path length or street congestion is minimized. The final case, called floating-track unidirectional routing, is presented in Chapter 5. Although the areas which may contain routes are limited in size as for the case considered in Chapter 4, existing routed paths may be reassigned to other tracks to accommodate new paths as necessary. Paths are routed such that either path length, street congestion, or the number of routed paths perturbed is minimized.

Chapters 6 and 7 consider two specific routing cases. In Chapter 6, the unidirectional configuration is uncon-

strained as in Chapter 3, but the cost function employed in finding a path is to minimize the number of switches between the line of points. The model used in this case is a derivative of the graph model used in Chapter 3. Chapter 7 considers the case where no switches are permitted in the unconstrained model. Thus, each two-point net is entirely routed in one or the other of the two sides of the line of points. A graphical method for optimally assigning a path to an area using bicolored nodes is described.

Chapter 8 presents the application of the Lee cellular algorithm in solving the same problem considered in Chapter 3. The Lee algorithm performs as well as our graphical method for the unconstrained case. However, it appears to be limited in its applicability to the other routing cases as presented in Chapters 4, 5, and 6.

Chapter 9 in this dissertation presents experimental results for several of the major algorithms given in Chapters 3 and 4. Basic conclusions and recommendations for further research are presented in Chapter 10.

CHAPTER 2

GENERAL DEFINITIONS

2.0 Introduction

The general notation and definitions relevant to unidirectional routing are presented in this chapter. These definitions generally apply to the entire thesis, but are particularly pertinent to Chapters 3, 4, and 5. Specific definition, as they apply to particular algorithms or concepts, are given in later chapters as needed. Previous efforts and results are also presented. The basic model used in Chapters 3-5, namely the access graph, is described. Finally, the minimum-cost path algorithm due to Dijkstra is presented.

2.1 Points

Let $a_1, a_2, \ldots, a_m$ be an ordered set A of m points ($m > 1$) lying on a horizontal straight line with end points a_1 and a_m. This line is called the point line.

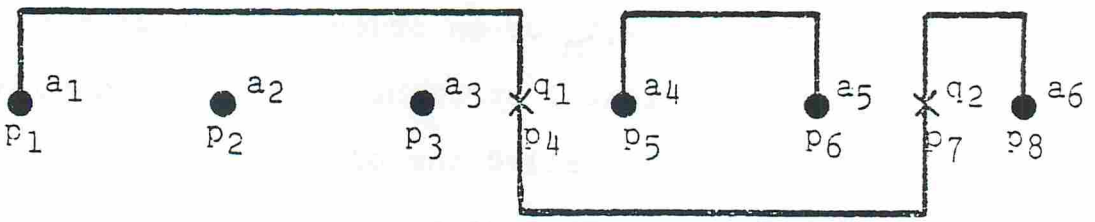
Each point a_i represents a pin or via on a printed circuit board and is called an *original point*. Unless otherwise stated, let the distance between adjacent points be of unit length, and let the location of point a_i be to the left of point a_{i+1}.

We further define an ordered set Q of *pseudo-points*,

$q_1, q_2, \ldots, q_r$, $r \geq 0$, that exist on the point line such that each point q_i represents the intersection of a routed path and the point line. Any two pseudo-points q_i and q_{i+1} are ordered in Q such that the location of q_i is to the left of q_{i+1}. The two point sets Q and A are mutually exclusive since $a_i \neq q_j$ for any $1 \leq i \leq m$ and $1 \leq j \leq r$.

Finally, we define an ordered set C of $n = m+r$ points, $p_1, p_2, \ldots, p_n$ (called a *unidirectional point set*) where $C = A \cap Q$, and the order of each point p_i is established by the left-to-right location sequence of pins, vias, and pseudo-points on the printed circuit board. Point p is located at α_i.

Figure 2.1 shows an example unidirectional configuration illustrating the point sets described above as well as their ordering.



Set $A = \{a_1, a_2, \ldots, a_6\} = \{p_1, p_2, p_3, p_5, p_6, p_8\}$

Set $Q = \{q_1, q_2\} = \{p_4, p_7\}$

Set $C = A \cup Q = \{p_1, p_2, \ldots, p_8\} = \{a_1, a_2, a_3, q_1, a_4, a_5, q_2, a_6\}$

Figure 2.1. An example configuration showing point sets A, Q, and C. Pseudo-points are shown as "x" on the point line.

2.2 Intervals

The space along the point line between each adjacent point p_i and p_{i+1} is called an *interval* u_i. To be precise, an interval u_i is the semi-closed interval $[p_i, p_{i+1})$ where p_i is contained within u_i, but p_{i+1} is not. The last interval u_n on the point line is defined to consist of only the point p_n. Let the length of interval u_i, denoted by $|u_i|$, equal $\alpha_{i+1} - \alpha_i$. Throughout this thesis intervals are often referenced by their leftmost point. Thus, interval p_i is synonomous with interval u_i.

2.3 Paths and Configurations

A *unidirectional configuration* D (or simply, *configuration*) is defined to consist of a point set P and two areas adjacent to and on opposite sides of the point line. These areas are called *streets*, and they contain the paths in D which interconnect points in P. The street located above the point line is the *upper* street S^+, and the street located below the point line is the *lower* street S^-. Paths are routed in horizontal and vertical *tracks*. Horizontal tracks are restricted to exist between the ends of the point line. Vertical tracks (called *switching tracks*) are perpendicular to horizontal tracks and extend from the upper edge of street S^+ to the lower edge of street S^-. Figure 2.2 illustrates the concepts of streets and tracks.

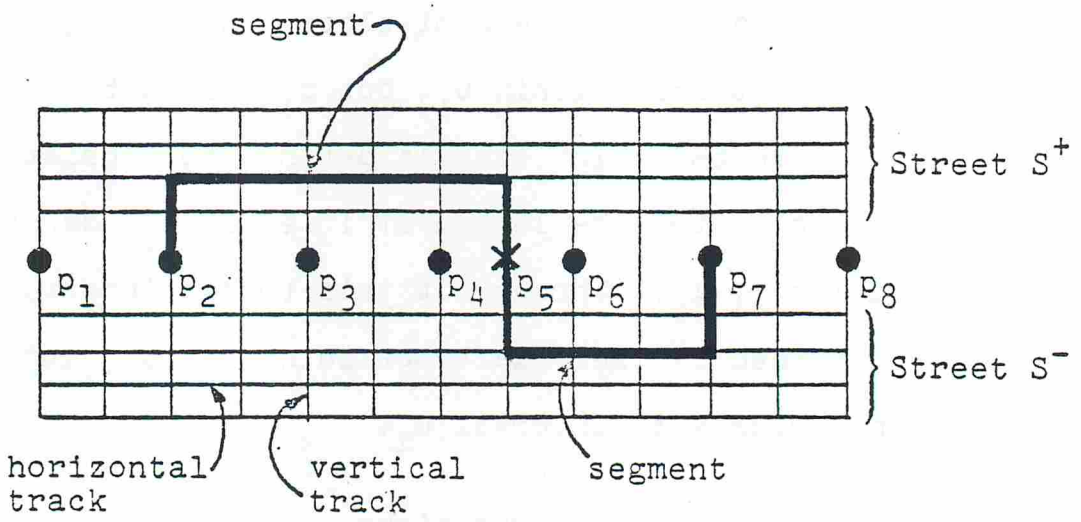


Figure 2.2. An example configuration illustrating the concepts of paths, streets, and tracks. A path containing one pseudo-point (p_5) and end points p_2 and p_7 is shown.

A *path* contains zero or more pseudo-points and two original points as its end points, and does not intersect any other path. A *feasible path* is a path which has not yet been assigned to tracks. A *routed path* is a path which lies entirely within tracks.

A *segment* (p_i,p_j) is a subpath which contains no points other than its end points p_i and p_j. If the segment is feasible its end points may be located within intervals (i.e., not necessarily attached to a point p_i). A segment

is located entirely in one of the two streets. For instance, in Figure 2.2, the subpath between points p_2 and p_5 is the segment (p_2,p_5). A segment is a *routed segment* if it is part of a routed path. Otherwise, it is a *feasible segment*.

The horizontal part of a segment is called a *section* whereas the vertical part of a path connecting two sections (and, for routed paths, defines a pseudo-point) is called a *switch*. Thus, a path consists of one or more sections connected by zero or more switches, and contains two short vertical *links* at either end of the path connecting the two points to sections. The path shown in Figure 2.3 has links at points p_2 and p_7, and sections and switches as shown. Similarly, segments, sections, switches, and links are considered to be either routed or feasible.

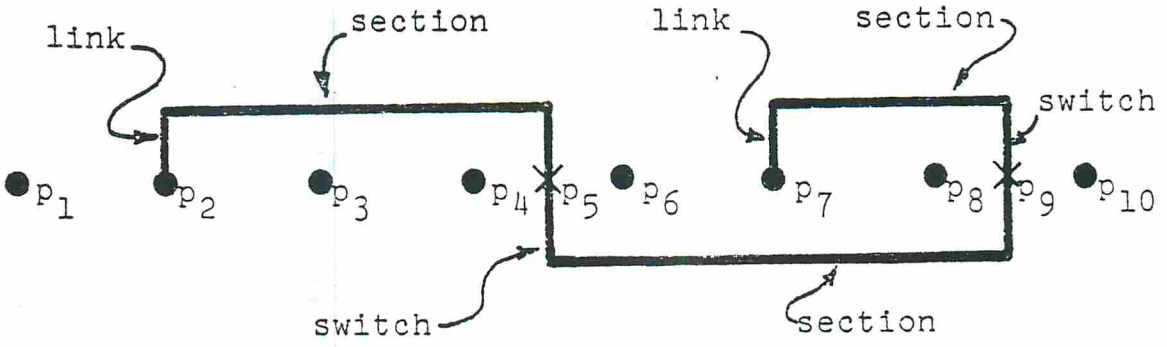


Figure 2.3. An example configuration illustrating the definition of sections, switches, and links.

The point at which a section and switch meet is called a *connection*. If only one section meets at a switch or link, then the connection is a *chain connection*. If two sections meet at a switch or link, then the connection is called a *tree connection* (see Figure 2.4). Tree connections are considered only in Chapter 7.

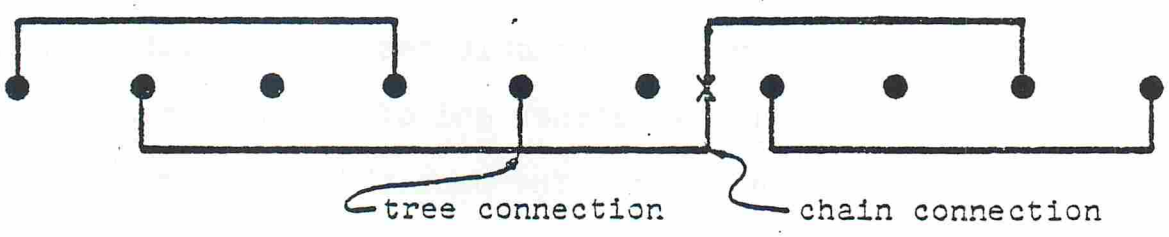


Figure 2.4. A configuration illustrating both tree connections and chain connections.

If a configuration contains a feasible path, it is a *feasible configuration*. Otherwise, it is a *routed configuration*. There must be no more than one path between any two points in a configuration.

If a point p_i in a configuration D is the end point of only one segment, then p_i is said to be *single-covered*. If there are two segments attached to p_i, then p_i is *double-covered*. Otherwise, point p_i is *uncovered*. By definition, a pseudo-point is double-covered.

2.4 Interval Accessibility

We define an interval u_i in a configuration D to be *accessible* to interval u_j $(i \neq j)$ if a path exists from interval u_i to interval u_j in a given street. For example, assuming there exists an ample number of free tracks, interval u_i is accessible to interval u_2 in street S^- of the configuration shown in Figure 2.5. Interval u_1 is likewise accessible to interval u_6 in street S^+; however, point p_1 is not accessible to interval u_6 in street S^+ since only one net can cover a point in a given street.

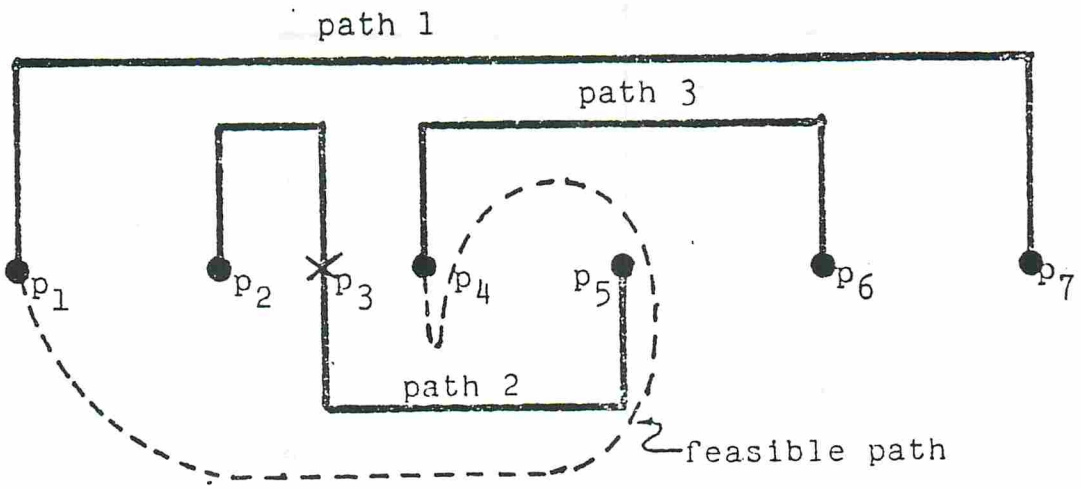


Figure 2.5. An example configuration illustrating accessible intervals. Feasible paths are shown as dotted lines in this and subsequent figures.

2.5 Channels

A channel is a rectangular area of the unidirectional configuration which contains no routed wires and is bounded by routed sections, the point line, or the edge of the configuration. There are two types of channels — section channels and switching channels (see Figure 2.6).

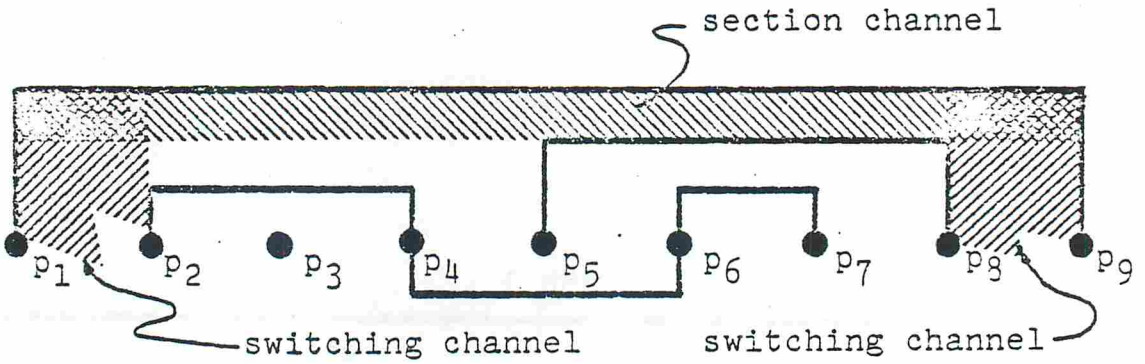


Figure 2.6. An example illustrating section and switching channels.

A *section channel* is bounded above by a routed section (called the *outer* section) or the edge of the configuration, bounded below by one or more routed sections (called the *inner* section(s)) or the point line, and bounded at either end by a link switch, or edge of configuration. Let a section channel extend between interval u_i and interval u_j. Then the *length* of the section channel is $\alpha_i - \alpha_j$ and the

width of the channel is the number of horizontal tracks it contains. Obviously, interval u_i is accessible to interval u_j if the channel width is greater than zero.

As shown in Figure 2.6, the width of a *switching channel* is the width of an interval and is bounded above and below by either a routed section or the configuration edge. The length of a switching channel is arbitrarily defined to be 1 throughout this work since our primary interest is in routine sections. The width of a switching channel is the number of switching tracks it contains.

Obviously, feasible sections are routed in section channels and feasible switches are routed in switching channels.

The concept of channels is central to this work since it is configuration channels which are modeled by arcs in the access graph. Furthermore, the label of each arc contains all the desired characteristics about the channel it represents such as the length and width of the channel. Thus, as will be seen later, a path in a configuration is defined as a path in the access graph for that configuration. The objective function for determining the path is defined on the channel characteristics as reflected in the access graph.

The definition of channels is further refined in Chapter 4.

To calculate path lengths, we consider a feasible switch to be arbitrarily close to the point in the interval containing the switch. The *length* of a path is defined to be the total length of each section in the path plus the number of switches in the path. The length of a section (p_i,p_j), $i < j$, is $\alpha_j - \alpha_i$ where α_i and α_j are the location of points p_i and p_j. For instance, if each interval in the configuration in Figure 2.5 is one unit except for intervals u_2 and u_3 which are each of length 1/2, then path 2 is of length

$$(\alpha_3-\alpha_2) + (\alpha_5-\alpha_3) + 1 = 1/2 + 1-1/2 + 1 = 3 \text{ units}.$$

The length of the feasible path shown in Figure 2.5 is

$$(\alpha_5-\alpha_1) + 1 + (\alpha_5-\alpha_4) + 1 = 3+1+1+1 = 6 \text{ units}.$$

2.6 Canonical Subpaths

The concept of canonical segments and sections is integral to the formation and use of the graphical models developed later. A *canonical segment* (abbreviated *c-segment*) is a feasible segment (p_i,p_j) such that there exists no interval u_k accessible to intervals u_i or u_j where $\alpha_i < \alpha_k < \alpha_j$, $(i < j)$. For example, the feasible segment (p_4,p_5) in street S^+ in the configuration shown in Figure 2.5 is

canonical whereas feasible segment (p_1,p_5) in street S^- is not.

A *canonical section* (abbreviated *c-section*) is the horizontal part of a c-segment. The concept of c-section sequences is key to the routing methods developed later. Every section is composed of a sequence of c-sections. For example, the feasible section (p_1,p_5) in the configuration shown in Figure 2.5 is composed of the sequence of c-sections (p_1,p_2) and (p_2,p_5).

A similar concept of canonical section channels is also apparent. A canonical section channel intersects switching channels only at its ends. For instance, in Figure 2.6 section channel (p_1,p_8) is not a canonical section channel, but section channel (p_4,p_8) is.

2.7 The Unidirectional Routing Problem

The unidirectional routing problem is one of determining a path between two given points $N = (p_i,p_j)$ in a routed configuration D which satisfies some cost function, and then assigning that path to appropriate tracks in D thus forming a routed path.

2.8 Previous Work

All previously reported methods for the solution of the unidirectional routing problem center around the heuristic method described in Ting [23]. A later report by

Kuh [14] presents a different but related approach without giving an algorithm for solution. Ting's method is described here.

Let set N be a set of nets to be routed in a unidirectional configuration and which have been assigned to streets. Let $\bar{N}$ be the remaining nets to be routed in the configuration but which are not assigned to streets. The assignment of segments to tracks is not performed. The assignment procedure given by Ting generates set N such that regions of high wire congestion in the board are reduced in both streets. The fact that a net is assigned to a street implies only that the end links are routed in the assigned street. The actual path for the net as found by a separate procedure may actually switch between streets. The results of the assignment procedure are not optimal although a claim is made that a path for a net will always be found if there exists a path. However, Kuh has retracted this claim [14] since counter examples have been shown to exist.

The street assignment procedure defines a theoretical minimum measure of routing congestion. Attempts are then made to allocate nets to streets to meet this measure. Should the theoretical minimum not be obtained, the measure is increased by one to aid the allocation. This process is repeated until no more nets can be allocated.

The path-finding procedure is simple and effective. Let L be the ordered list of nets n_i to be routed ($L = N \cup \bar{N}$). Double-covered points are not permitted in Ting's model (i.e., only two-point nets are routed). The procedure is to take each net in turn from L and route it such that the end points are covered in the assigned street, and the path segments traverse each point in the street opposite to that in which the point is covered. For points that are uncovered, the path segment traverses the upper street. For instance, net γ is routed as shown in Figure 2.7.

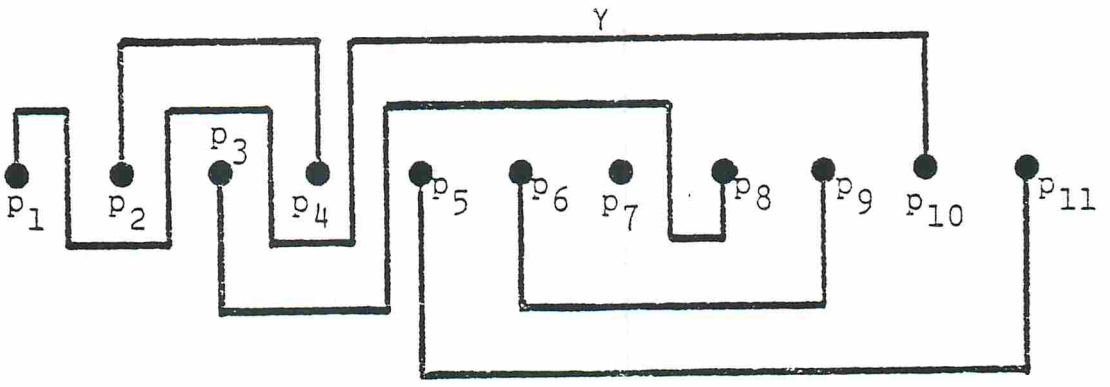


Figure 2.7. Example of routing using Ting's method.

The constraints and results given in this thesis differ from those given by all previous reported results,

including Ting's, in the following ways.

1. All results given in this thesis are optimum with respect to the stated cost goals. No previously reported results are optimal in any sense.

2. The unidirectional model considered by other researchers only accommodates two-point nets. The model considered in this thesis accommodates multipoint nets defined on the point line.

3. Routability is not guaranteed by other researchers. All methods considered in this thesis guarantee that a path will be found if a path exists.

4. Street capacities are forced to be the same between the upper and lower streets in the models considered by other researchers. No such restriction is imposed here.

Finally, a complexity of $O(n^2)$ is quoted for Ting's method while exponential complexity is required for Kuh's method [14].

2.9 A Graphical Approach

The method of solution presented in this thesis requries that a graph G (called an access graph) be generated from D. G reflects all switching channels and canonical section channels in D. The solution steps for routing a

net N in D are:

1. given D, generate G.
2. obtain a path in G meeting the desired cost function. This path represents a feasible path for N in D.
3. assign the feasible path to tracks thus obtaining a routed path for N in D.

Based upon track constraints in D, the unidirectional routing problem can be viewed in three possible ways. First, the number of tracks is unlimited. For this case we ignore the existence of tracks when finding paths. This problem is called *unconstrained routing*, and is considered in detail in Chapter 3.

In the second problem type, called *fixed track routing*, the number of available tracks is fixed with total capacity T. Once a feasible path is assigned to tracks, it remains fixed to these tracks as additional paths are routed. This routing problem is considered in detail in Chapter 4.

Finally, we can consider the case where the number of tracks is limited to some capacity T, as for the fixed-track case, but routed paths may be reassigned to alternate tracks, if necessary, to free a track for routing a new feasible path. This routing problem is called *floating-track routing*, and is considered in detail in Chapter 5.

2.10 The Access Graph

We now consider a graphical model of a routed configuration D which is useful in the solution of the problem of unidirectional routing. Essentially, the nodes of the graph represent intervals, and the arcs represent section and switching channels in D. The detailed algorithms that generate appropriate access graphs for each of the unidirectional routing problems are presented in the next three chapters.

Let D be a routed configuration with n points. Let G be an *access graph* $G = (V,E)$ associated with D, where V is a set of 2n nodes and E is a set of undirected arcs defined on the nodes in V. The node set V consists of two mutually exclusive subsets V^+ and V^- each consisting of n nodes. A node v_i^+ in V^+ represents the S^+ street side of the interval u_i while a node v_i^- in V^- represents the S^- street side of interval u_i. A node v_i^+ (v_i^-) in G is said to be *covered* if point p_i in D is covered in street S^+ (S^-). Thus a node in G is either covered or uncovered.

The set E consists of two types of arcs, namely section arcs and switching arcs. Let x be one of the two symbols $\{-,+\}$. A *section arc* (v_i^x, v_j^x) exists between two nodes v_i^x and v_j^x $(i \neq j)$ where v_i^x and v_j^x are in V^x if interval u_i is accessible to interval u_k between points p_i and p_j such that interval u_k is accessible to intervals u_i or u_j in

street S^x. Thus arc (v_i^x, v_j^x) represents a canonical section channel between intervals u_i and u_j in D. Each section arc in G is labeled as (x,B), where set B is a set of values reflecting characteristics of the associated channel in D such as the length of the channel, the maximum routed-path congestion over the length of the channel, etc.

A *switching arc* $(v_i^x, v_i^{\bar{x}})$ exists between every two nodes v_i^x and $v_i^{\bar{x}}$ where $v_i^x \in V^x$ and $v_i^{\bar{x}} \in V^{\bar{x}}$ ($\bar{x}$ is the complement of x) if a new switch can be routed between points u_i and u_{i+1} in D. v_i^x and $v_i^{\bar{x}}$ represent the same interval u_i. An exception occurs for interval u_n since point p_{n+1} does not exist, and by definition, no paths may be routed to the right of point p_n. Thus, no switching arc exists between nodes v_n^x and $v_n^{\bar{x}}$. A switching arc $(v_i^x, v_i^{\bar{x}})$ represents a switching channel in interval u_i. A switching arc is labeled ("w",B) where B is a set of values similar to those for section arcs.

To clarify the definition of the access graph, consider the configuration D shown in Figure 2.8. Assume there are sufficient unassigned tracks where necessary. The length of each interval is assumed to be 1 unit with the exception of intervals 5 and 6 which are each 1/2 units long. Point p_6 is a pseudo-point.

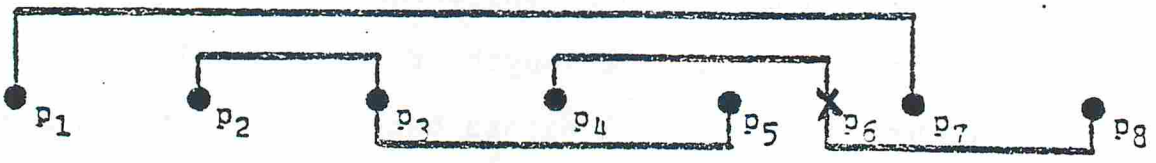


Figure 2.8. An example configuration D illustratting the definition of the access graph G.

Assume we are only interested in the lengths of paths. Then the label on each arc is the pair (x,δ) if the arc is a section arc, or $(w,1)$ if the arc is a switching arc, where x is "+" or "-", δ is the length of the section channel, and the length of each switch has been arbitrarily defined to be 1 unit. All of the section channels in configuration D are listed in Table 2.1. The channels identified by an asterisk are not canonical and thus do not appear as arcs in Figure 2.9. We use the left point p_i of an interval u_i to represent that interval in the table. Interval u_8 consists of only point p_8.

Table 2.1. All Section Channels in the Configuration shown in Figure 2.8.

Street S^+	Street S^-
(p_1,p_3)	(p_1,p_2)
$*(p_1,p_6)$	$*(p_1,p_5)$
(p_3,p_6)	$*(p_1,p_8)$
(p_7,p_8)	(p_2,p_5)
(p_4,p_5)	$*(p_2,p_8)$
	$*(p_5,p_8)$
	(p_3,p_4)
	(p_6,p_7)

The access graph G for the configuration shown in Figure 2.8 is shown in Figure 2.9. Each canonical channel listed in Table 2.1 is reflected as an arc in G as are the seven switching arcs possible for this example (there is no switching arc for interval p_8). The length of each section arc (p_i,p_j) is $|\alpha_j-\alpha_i|$, and it is assumed all channels have width greater than zero.

The utility of the access graph is evident by picking any two nodes, say nodes 1^- and 5^+, and determining a path between them in G. Suppose we choose path $(1^-,2^-,5^-,5^+)$ in Figure 2.8. This corresponds to a feasible section in configuration D from point p_i to interval u_5 in the lower street and a switch in interval u_5. The feasible path is

completed by two links, one at point p_1 and the other at point p_5, and a short section from point p_5 to interval u_5. The sum of the lengths of the arcs in the chosen path is 5 units which is also the length of the feasible path in D.

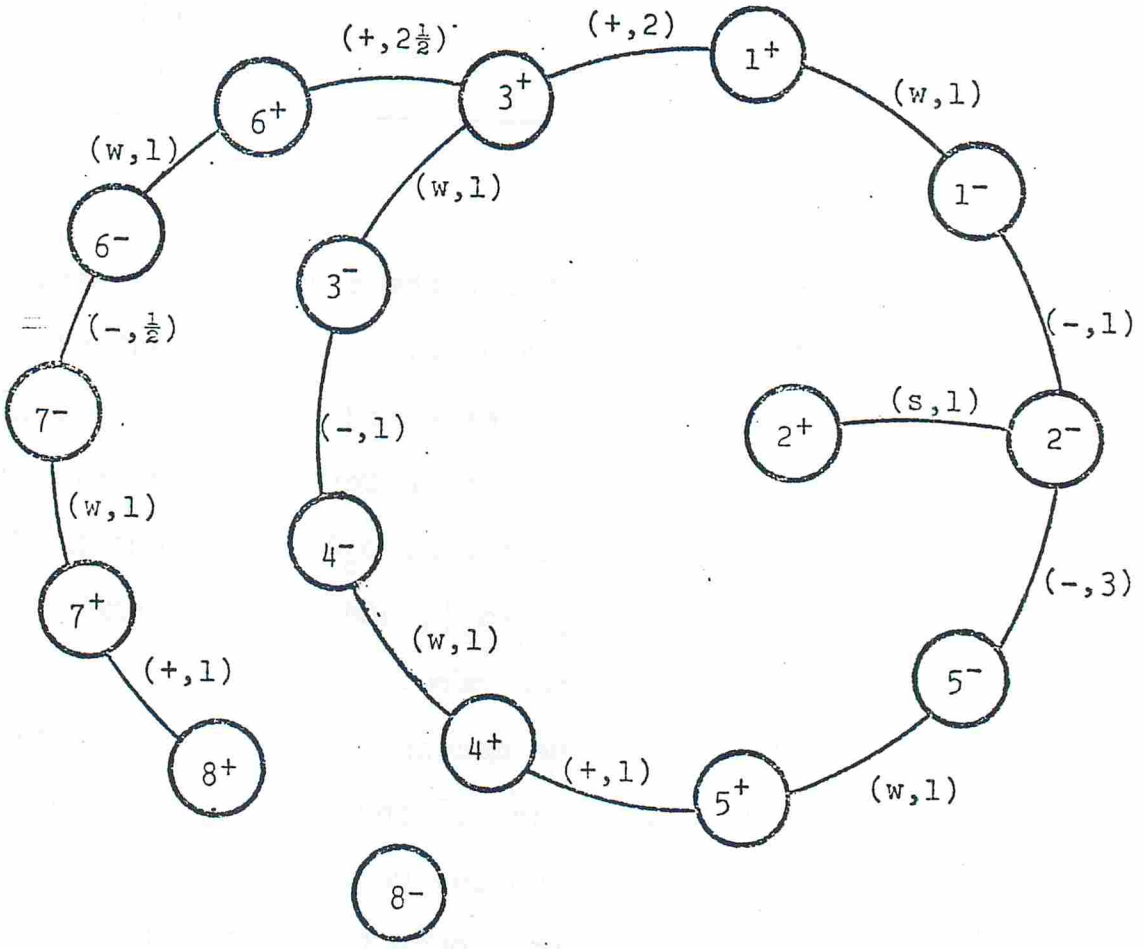


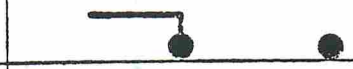
Figure 2.9. The access graph G for the configuration shown in Figure 2.8.

There is another path between nodes 1^- and 5^+, namely $(1^-,1^+,3^+,3^-,4^-,4^+,5^+)$. The length of this path is 7 units. If we want to route shortest paths in D, then the path of length 5 units is chosen as the desired one.

2.11 Adjacent Point Configurations

It is interesting to note that there are a total of nine adjacent point configurations possible on two points as observed from a given street. These configurations are shown in Table 2.2. It is from the relational properties of adjacent point configurations that the access graph algorithms in Chapters 3, 4, and 5 are derived.

Table 2.2 Table of All Adjacent Point Configurations on Two Adjacent Points

Configuration	P_i P_{i+1}
1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	
9.	

2.12 A Least-Cost Path Algorithm

The routing methods presented in this work are based upon determining a least-cost path between two nodes in an undirected weighted graph. The least-cost path algorithm presented here is due to Dijkstra [24]. This algorithm was selected for use since it operates in time $\mathcal{O}(e\log\eta)$ for a graph with e edges and η nodes. Most other least-cost path algorithms operate in time $\mathcal{O}(\eta)$ (see Rubin [25], Bondy [26]). Since for most graphs considered in this work $e << 3\eta$, and $e_{max} = 3\eta-1$, Dijkstra's algorithm shows a distinct advantage over other least-cost path algorithms.

The algorithm outlined here determines the shortest distance from a given source node u_0 to all other nodes of G. Let S_i be a proper subset of V, the node set of G, such that $u_0 \in S_i$, and let $\bar{S}_i$ be the subset of V which consists of all nodes not in S_i. Let V contain η nodes. The basic idea is to construct an increasing sequence $S_0, S_1, \ldots, S_{\eta-1}$, of subsets of V in such a way that, at the end of stage i, shortest paths from u_0 to all nodes in S_i are known. We further define a label $\ell(u_j)$ for each node u_j. $\ell(u_j)$ will be the cost (i.e., the "length") of a least-cost path from u_0 to u_j when the algorithm has processed set S_i where $j \geq i$. $w(u,v)$ denotes the weight on arc (u,v).

Dijkstra's Algorithm

1. Set $\ell(u_0) = 0$, $\ell(v) = \infty$ for $v \neq u_0$, $S_0 = |u_0|$ and $i = 0$.

2. For each $v \in \bar{S}_i$, replace $\ell(v)$ by $\min\{\ell(v), \ell(u_i) + w(u_i, v)\}$. Compute $\min_{v \in \bar{S}_i} \{\ell(v)\}$ and let u_{i+1} denote a node for which this minimum is attained. Set $S_{i+1} = S_i \cup \{u_{i+1}\}$.

3. If $i = \eta - 1$, stop. Otherwise, replace i by i+1 and go to step 2. □

The timing of Dijkstra's algorithm is determined primarily by the two operations in step 2:

(1) ... replace $\ell(v)$ by $\min\{\ell(v), \ell(u_i) + w(u_i, v)\}$.

(2) Compute $\min_{v \in \bar{S}_i} \{\ell(v)\}$ ... .

Operation (1) must be executed for each node $v \in \bar{S}_i$ adjacent to node u_i. Thus, if d(u) is the degree of node u_i, then operation (1) is executed

$$\sum_{i=0}^{k} d(u_i) \qquad (1)$$

times after $k \leq \eta - 1$ iterations of the algorithm have occurred. Since the number of edges e in a graph G is equal to 1/2 the total degree of G, then operation (1) is executed a maximum of 2e times if Dijkstra's algorithm is run to completion (i.e., $k = \eta - 1$).

The number of times operation (2) is executed is a little more complicated to determine. Note that the size

of $\bar{S}_i$ is η-i-1; i = 0,1,2,...,η-1. If a linear search is performed to find the minimum $\ell(v)$ then operation (2) is executed a total of $\sum_{i=0}^{k} (\eta-i-1)$ times after $k \leq \eta-1$ iterations of step 2 in the algorithm. A more efficient method of determining the minimum element in set $\bar{S}_i$ is to sort $\bar{S}_i$ every time operation (2) is executed. If only a few elements in $\bar{S}_i$ differ from those in $\bar{S}_{i-1}$, then some efficiency can be gained if the nodes in $\bar{S}_i$ are structered as a balanced heap. Initially $(\eta-1)\log(\eta-1)$ operations are required to create a heap of $\eta-1$ nodes. Every subsequent iteration only requires $\log(\eta-i-1)$ operations to restructure the heap. Thus, no more than

$$(\eta-1)\log(\eta-1) + \sum_{i=0}^{k} d(u_i)\log(\eta-i-1) \qquad (2)$$

operations are required for operation (2).

From both expressions (1) and (2) and noting that $\eta+2$ and $2(\eta-1)$ operations are required for steps 1 and 3 of Dijkstra's algorithm, we see that the total number of operations $\mathcal{O}ps_k$ for the algorithm up to iteration k is

$$\mathcal{O}ps_k = 3\eta + (\eta-1)\log(\eta-1) + \sum_{i=0}^{k} [d(u_i)(1+\log(\eta-i-1)]. \quad (3)$$

Since $d(v) \leq 3$ for any node v in an access graph G, and assuming there are n unidirectional points in D, then

$\eta = 2n$. Thus, any least-cost path of length k in G representing D may be found with no more than

$$\mathcal{O}ps_k = 6n+3k+3+(2n-1)\log(2n-1)+3\sum_{i=0}^{k}\log(2n-1-i) \qquad (4)$$

operations. If the least-cost path in G contains every node in G, then

$$\mathcal{O}ps_n = 12n+(2n-1)\log(2n-1)+3\sum_{i=0}^{2n-1}\log(n+i) \qquad (5)$$

operations are required.

The access graph which maximizes the node degree of the graph is the one for an empty configuration. All nodes except those representing points p_1 and p_n in D have degree 3. The maximum path length in such a graph is $k = n$ and occurs for a path in a street in D from points p_1 to p_n. For this case the first node in the path has degree 2, the last node has degree 1, and the other n-2 nodes have degree 3. Thus,

$$\mathcal{O}ps_{max} = 6n+(2n+1)\log(2n-1)+\log(n)+3\sum_{i=0}^{n-1}\log(2n-i-1) .$$

In contrast to this maximum case, the minimum number of operations required by Dijkstra's algorithm occurs when the path in D is desired between two adjacent points and one of the points is p_n. For this case $k = 2$, the minimum degree of node v_{n-1} is 2, and the degree of node v_n is 1. Then,

$$\mathcal{O}ps_{min} = 6n+(2n+2)\log(2n-1)+3\log(2n-2)$$

$$\approx 6n+(2n+5)\log(2n-1).$$

In all cases, the complexity of Dijkstra's algorithms as applied to the problems contained in this thesis is $\mathcal{O}(n\log n)$.

CHAPTER 3

UNCONSTRAINED UNIDIRECTIONAL ROUTING

3.0 Introduction

The simplest unidirectional routing problem to solve is routing a two-point net N between two points in a configuration D ignoring track assignments. We are interested in finding a feasible path for N in D where the cost function we choose to minimize is path length. First, an algorithm is presented which generates the unconstrained access graph G for a configuration D. Next, an analysis of both the algorithm and the graph produced by the algorithm is given. Then a brief discussion on finding the minimum-length path in G is given. Finally, this path is directly mapped into a feasible path in D.

3.1 The Access Graph for the Unconstrained Routing Problem

Algorithm 3.1 generates an access graph G for a configuration D where tracks are ignored. Each point p_i, $i = 1,2,\ldots,n$, is examined in turn, first from the viewpoint of street S^+ and then from street S^-. At each point p_i, the algorithm pops the top value (say, u_j) from a stack S if interval u_i is accessible to interval u_j, and u_j is to the left of u_i. Likewise, i is pushed onto stack S if in-

terval u_i is accessible to some interval u_j where u_j is the first accessible interval to the right of u_i. Whenever S is popped an arc is constructed between nodes u_i^x and u_j^x (x identifies the street being processed). The jump table in step 4 is derived from the adjacent point configurations shown in Figure 2.8.

Algorithm 3.1. Construction of the unconstrained access graph

Purpose: This procedure generates the access graph of a given (routed) configuration D where street capacities and track assignments are ignored.

Method: The line of configuration points is scanned twice, from left to right, once for street S^+ and a second time for street S^-. During each scan, successive adjacent points p_i and p_{i+1} are examined with respect to routed path coverings and the directions of the paths covering them. Appropriate action is taken (via the table in step 4) based upon which of the nine adjacent point configurations shown in Figure 2.9 is applicable. Algorithm steps 5 to 8 perform stack and graph operations such that at the conclusion of the second point scan, the entire access graph is constructed. The procedure operates on one stack, S, which contains items j, where p_j is a point in D. We define point p_{n+1} to be an uncovered imaginary point lying just to the right of point p_n on the point line in D.

Input: Routed configuration D including the location α_i of each point p_i.

Output: The unconstrained access graph G for D.

Procedure:

Step 1. (initialization) Set x = "+", and clear stack S. Construct the 2n nodes $1^+, 2^+, \ldots, n^+$, $1^-, 2^-, \ldots, n^-$ in G.

Step 2. (switching arcs) For each i = 1,2,...,n-1, construct an arc between nodes i^+ and i^-, and label it ("w",1).

Step 3. Set i = 0. Push i on stack S. (Even though there is no point p_0 in D, we need this dummy item in S to ensure correct processing of the end points in D.)

Step 4. (main loop) Let the top item in stack S be j. Set i = i+1. Consider points p_i and p_{i+1} in D. Select the applicable case from Table 3.1 and branch to the particular step as directed.

Step 5. Pop stack S to get j. If $j \neq 0$ then construct an arc in G between nodes i^x and j^x and label it $(x, \alpha_i - \alpha_j)$. Push i on stack S and go to step 8.

Step 6. Push i on stack S and go to step 8.

Step 7. Pop stack S to get j. Construct an arc in G between nodes i^x and j^x, and label it $(x, \alpha_i - \alpha_j)$. Go to step 8.

Step 8. (end of main loop) If $i \neq n$ then go to step 4.

Step 9. If x = "+" then set x = "-", set stack S empty, and go to step 3. Otherwise, exit procedure. □

3.1.1 Example of operation of Algorithm 3.1

It is instructive to consider an example of the use of Algorithm 3.1. Consider the configuration shown in Figure 3.1 and let each interval be of unit length except for intervals u_9 thorugh u_{12} which are each of length 1/2.

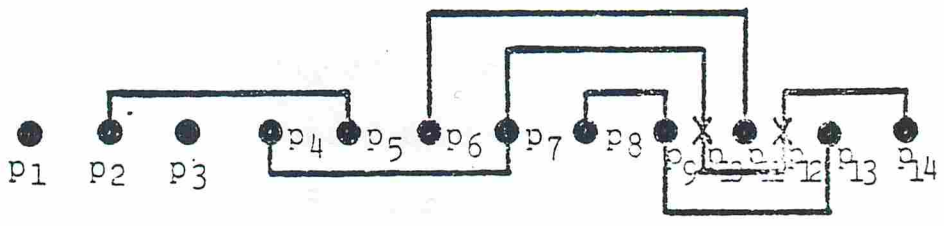


Figure 3.1. Routed configuration illustrating the use of Algorithm 3.1.

Table 3.1. The Adjacent Point Configuration Jump Table

Adjacent Point Configuration (Table 2.2)	p_i is Covered in S^x	p_{i+1} is Covered in S^x	p_i is Left Point of Segment	p_{i+1} is Left Point of Segment	Go to Step
1	no	no	—	—	5
2	yes	no	no	—	5
3	no	yes	—	yes	5
4	no	yes	—	no	7
5	yes	yes	no	no	7
6	yes	yes	no	yes	5
7	yes	no	yes	—	6
8	yes	yes	yes	yes	6
9	yes	yes	yes	no	8

The algorithm begins with x = "+", i = 1, and 0 on top of stack S. The access graph initially consists of 2n isolated nodes. The table in step 4 directs control to step 5 since point 1 is isolated, point 2 is covered in street S^+, and point 2 is the left-most point of segment (2,5). Since the top item in stack S is 0, we pop S and push i = 1 on S in step 5. In step 8, i < n so we update i = 2 in step 4 and jump to step 6 since point 2 is covered, point 3 is isolated, and point 2 is the left-most point of segment (2,5). We push i = 2 on stack S in step 6 and jump back to step 8. At this point, i = 3. We then enter step 5 since both points 3 and 4 are isolated. In this case, the top of stack S is 2 and thus we construct an arc between nodes 3^+ and 2^+ in the graph with label (+,1) since the length of interval 2 is 1 unit. The reader is encouraged to continue applying Algorithm 3.1 to this example until confident of its operation. Table 3.2 shows the sequence of steps taken by Algorithm 3.1 while processing the street S^+ side of the routed configuration shown in Figure 3.1. The stack figures in Table 3.2 refer to Figure 3.3. Figure 3.2 shows the complete access graph for this example.

3.1.2 Execution time of Algorithm 3.1

Consider unit operations to be:

1. construct an arc or node.
2. pop, push or clear stack S.

Table 3.2. The steps followed by Algorithm 2.1 in generating the access graph for the configuration in Fig.3.1. Only street S^+ processing is shown.

i	Process Step	Stack Action	Stack Fig.(3.3)	Arc Created	Label of Arc
1	5	Pop 0,Push 1	(a)		
2	6	Push 2			
3	5	Pop 2,Push 3	(b)	$(2^+,3^+)$	(+,1)
4	7	Pop 3		$(3^+,4^+)$	(+,1)
5	5	Pop 1,Push 5		$(1^+,5^+)$	(+,4)
6	6	Push 6			
7	6	Push 7			
8	8	none	(c)		
9	7	Pop 7		$(7^+,9^+)$	(+,2)
10	7	Pop 5		$(6^+,10^+)$	(+,3.5)
11	5	Pop 5,Push 11	(d)	$(5^+,11^+)$	(+,5)
12	6	Push 12			
13		Pop 12		$(12^+,13^+)$	(+,.5)
14	5	Pop 11,Push 14	(e)	$(11^+,14^+)$	(+,2)

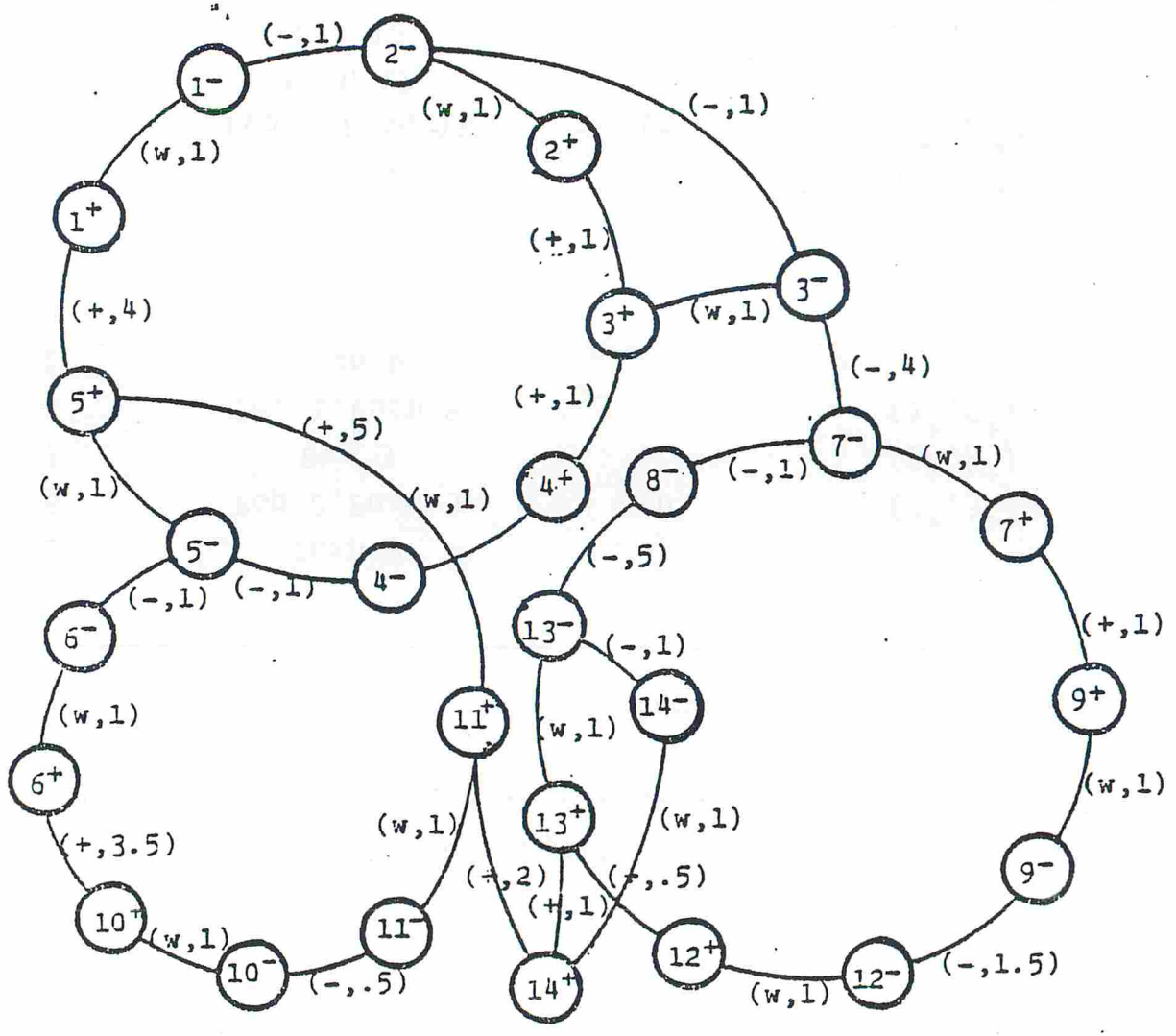


Figure 3.2. The access graph for the routing configuration shown in Fig.3.1.

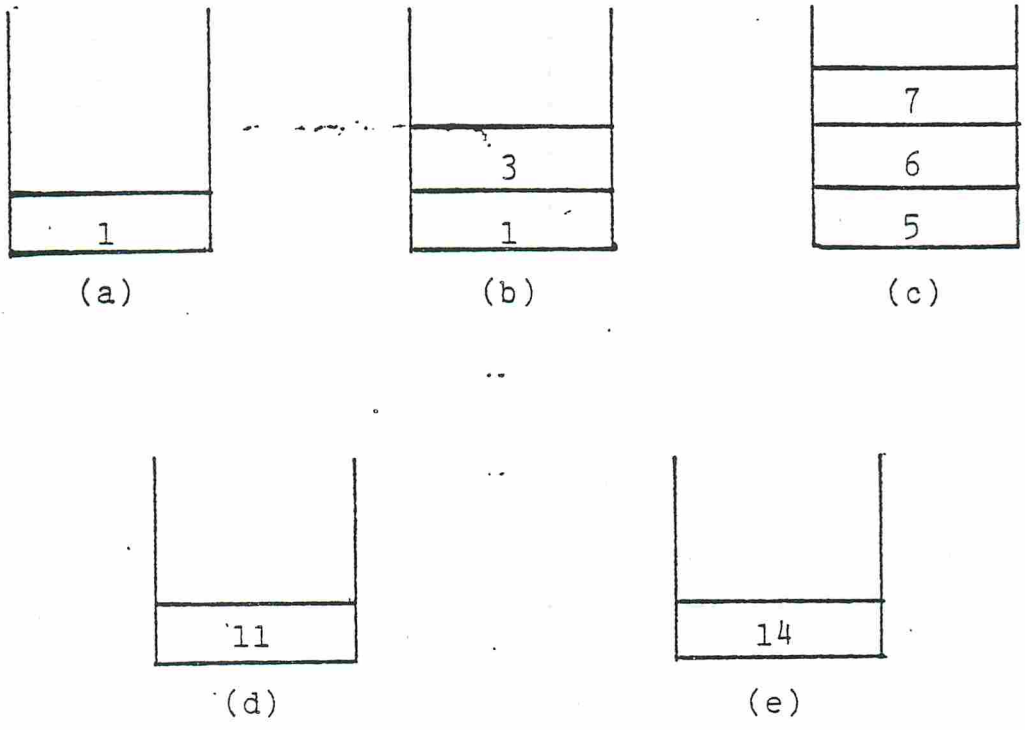


Figure 3.3. Stack contents for example of operation of Algorithm 3.1.

3. add or subtract two values.

4. test.

Each step in Algorithm 3.1 requires the number of operations shown in Table 3.3. Except for the initial steps of the algorithm, j does not equal zero. The maximum and minimum number of times steps 5, 6, and 7 are executed is configuration dependent. To find the number of times each

Table 3.3. Number of operations per step in Algorithm 3.1 as a function of the number of unidirectional points n.

Step	No. of Operations	No. of Times Step is Entered
1	2n+2	1
2	2n-2	1
3	2	2
4	2	2n
5	4	see discussion
6	1	see discussion
7	3	see discussion
8	1	2n
9	2	2

step is executed, consider the *precedence graph* G_p shown in Figure 3.4. This graph is obtained by letting each node represent one of nine routing configurations on two adjacent points in a configuration as shown in Figure 2.9, and constructing a directed arc from node i to node j if adjacent point configuration i can immediately precede adjacent point configuration j in some unidirectional configuration. Each node in G_p is labeled with an s or a t if that node represents an adjacent point configuration that could potentially be the first (s) or last (t) interval in a unidirectional configuration. Also, each node is labeled with an "f-value" of +1 if the left point of the adjacent point configuration represented by the node is the left end

point of a segment, an f-value of -1 if the right point of the adjacent point configuration represented by the node is the right point of a segment, or an f-value of 0 if neither of the above cases hold, or if both cases hold (e.g., adjacent point configuration 9).

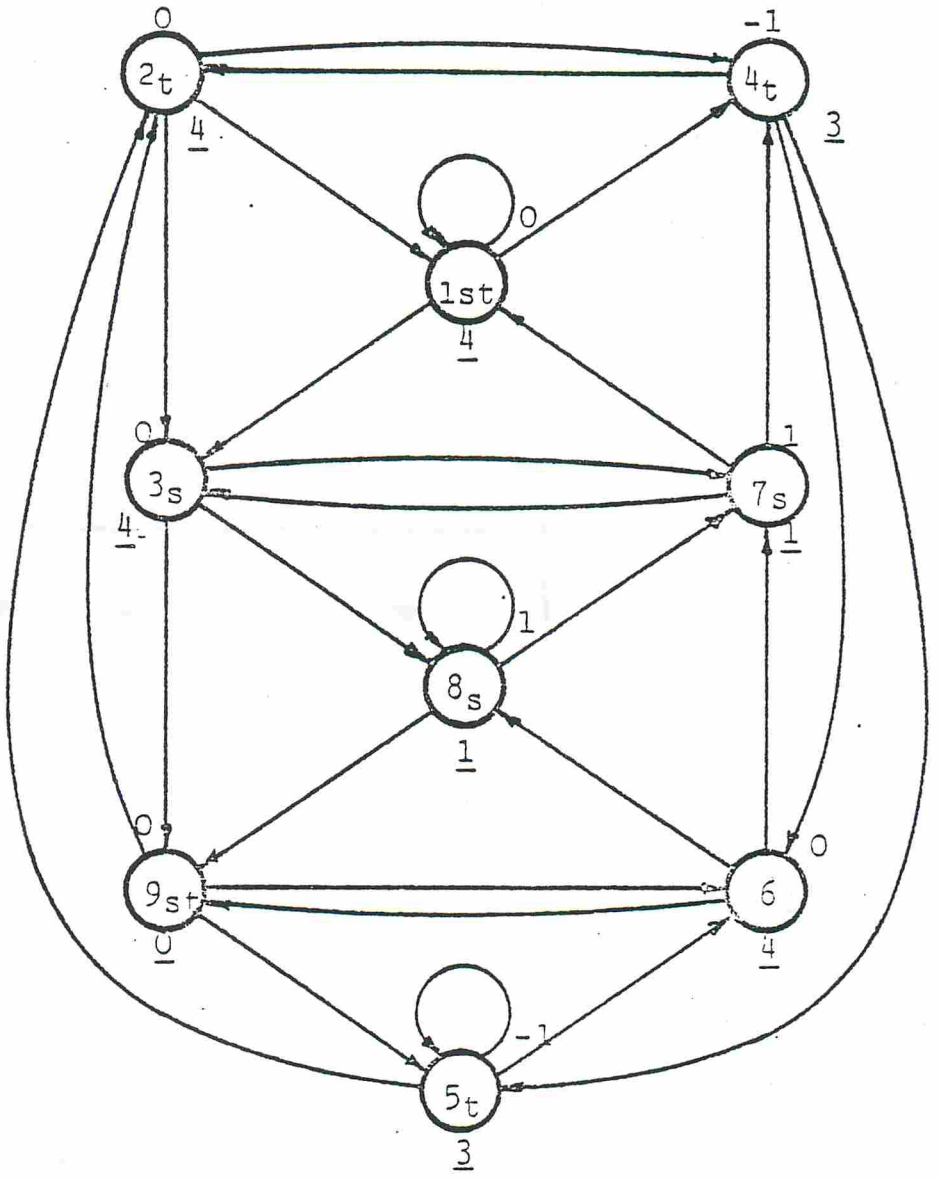


Figure 3.4. The precedence graph for the nine possible adjacent point configurations on two points.

The underlined numbers adjacent to each node in Figure 3.4 are the number of operations required by steps 5, 6, or 7 of Algorithm 3.1 to process the adjacent point configuration represented by each node.

The utility of graph G_p is readily seen by considering the example unidirectional configuration shown in Figure 3.5. As the configuration is scanned from left to right, a path is traced out in the precedence graph G_p.

node entered	3	7	3	9	5	2	1
running sum of f-values	0	1	1	1	0	0	0
running sum of Algorithm 2.1 operations (steps 5, 6, and 7 only)	4	5	9	9	12	16	20

Figure 3.5. An example unidirectional configuration illustrating the use of the precedence graph G_p to evaluate the number of operations required in steps 5,6, and 7 of Algorithm 3.1 when constructing an access graph from this configuration.

As can be seen in Figure 3.5, the number of operations required by steps 5, 6, and 7 of Algorithm 3.1 to construct an access graph from the configuration shown is 20. The number of operations per point is 20/8. Also note that the final sum of f-values over all nodes entered in G_p is 0. By observing these facts, we arrive at the following constraint rules for using the precedence graph G_p to create valid unidirectional configurations.

1. The sum of all f-values in the node sequence representing a valid unidirectional configuration must be zero. This occurs because the f-values actually keep track of the beginning and ends of segments.

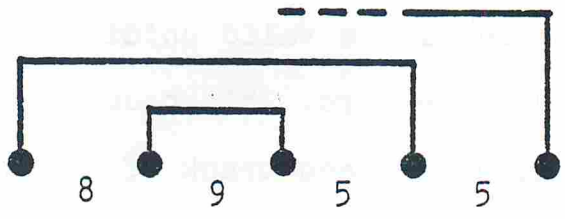
2. The cumulative sum of all f-values up to any intermediate point in the sequence of nodes representing a valid unidirectional configuration must be greater than or equal to zero.

3. The first node in the sequence of nodes representing a valid unidirectional configuration must be an s node and the last node must be a t node.

For example, the node sequence 8 9 5 represents a valid configuration (Fig. 3.6 a) since the above three rules are followed, whereas the node sequence 8 9 5 5 does not represent a valid configuration (Fig. 3.6 b) since both rules 1 and 2 are violated even though a path exists in G_p with

the node sequence 8 9 5 5.

(a) The valid unidirectional configuration for the node sequence 8 9 5.



(b) The invalid unidirectional configuration for the node sequence 8 9 5 5.

Figure 3.6. Examples of valid and invalid unidirectional configurations as characterized by graph G_p.

It is possible to simplify graph G_p by combining nodes with the same f-value together into common nodes with the sole exception of node 9. Thus, we create a new graph, G'_p, with four nodes {A,B,C,D} where A = {1,2,3,6}, B = {7,8}, C = {4,5}, and D = {9}. By collapsing nodes in G_p as just defined and inserting a directed arc in G'_p between two

nodes X and Y if at least one arc of the same direction exists in G_p between two nodes x, y, where $x \in X$ and $y \in Y$, the graph G'_p as shown in Figure 3.7 is constructed.

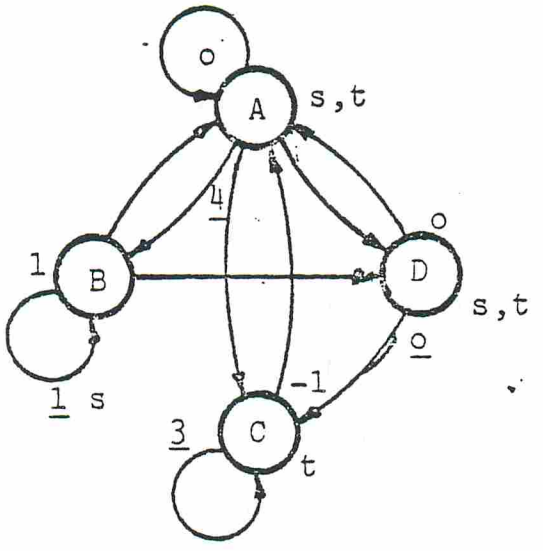


Figure 3.7. The simplified precedence graph G'_p.

Obviously, a path P in G_p maps into a path P′ in G'_p. If P contains n nodes, and satisfies the three constraint rules for valid unidirectional configurations, then P′ also contains n nodes and likewise satisfies those same rules. Also, the sum of the number of operations in Algorithm 3.1

in producing path P is the same as that to produce path P'. Thus, for our purposes, graph G'_p may be used to ascertain the minimum and maximum number of operations required by steps 5, 6, and 7 of Algorithm 3.1. The unidirectional configurations requiring those times is also inferred by G'_p.

First, consider the maximum number of operations. Observing graph G'_p, it is obvious that the path consisting only of a sequence of nodes AAA... gives a maximum number of operations of 4m, where m is the number of intervals in the unidirectional configuration producing this max value. If n is the number of unidirectional points, then n = m+1 and there are 4(n-1) operations required for one scan of the configuration. Determining the minimum number of operations is only slightly more difficult. Consider node D with an operation count of 0. Note that for a configuration of more than two points, node D may occur no more than 1/2 times the total number of nodes in any path. Also, nodes B and C must always occur the same number of times in any path satisfying the constraint rules. If we use the notation A^k to mean a node sequence of k A's where A^0 means node A is not in the sequence, then, by observation, we see that the sequence giving the smallest number of operations is of the form

$$B^k(AD)^mC^k \quad \text{or} \quad B^k(DA)^mC^k, \quad k \geq 0, \quad m \geq 0 .$$

The number of operations for either of these sequences is

$$3k + 4m + 0m + k = 4(m+k) .$$

The number of intervals in either of these configurations is

$$k + 2m + k = 2(m+k) .$$

Thus, the minimum number of operations per interval is $\frac{4(m+k)}{2(m+k)} = 2$ and the total number of operations for n points is 2(n-1) for one scan of the point line by the algorithm.

An example of the unidirectional configurations requiring the minimum and maximum number of operations by steps 5, 6, and 7 of Algorithm 3.1 is shown in Figure 3.8.

(a) The configuration requiring a maximum of four operations per point.

(b) A configuration requiring a minimum of two operations per point.

Figure 3.8. Example of configurations requiring minimum and maximum running times for steps 5, 6, and 7 of Algorithm 3.1.

Using the results just obtained, and Table 3.3, the minimum number of operations $\mathcal{O}ps_{min}$ and the maximum number of operations $\mathcal{O}ps_{max}$ are given by the expressions

$$\mathcal{O}ps_{min} = (2n+2) + (2n-2) + 4n + 4(n-1) + 2n+4 = 14n+4$$

and

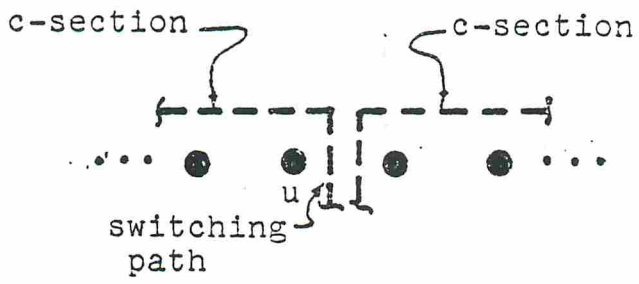
$$\mathcal{O}ps_{max} = 2n+2 + 2n-2 + 4n + 8(n-1) + 2n+4 = 18n .$$

Hence, the unconstrained access graph can be processed in linear time.

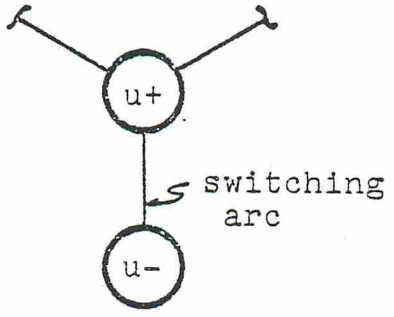
3.2 Bounds on the Size of the Unconstrained Access Graph

The size of G is important since algorithms which process G run in time proportional to the size of G. Therefore, we wish to get good bounds on the size of G. By definition, the number of nodes in G is fixed at 2n where n is the number of points in the configuration D represented by G.

It is interesting to note that the maximum degree of any node in G is three. This can be easily seen by observing that a feasible c-section may be drawn from the left or the right of any interval in a configuration D, and a switching path may be drawn in G between the two nodes representing that interval. These three items are represented by three arcs in G attached to the same node. Figure 3.9 illustrates this point.



(a) Part of configuration D showing all possible paths in interval u.



(b) Subgraph of G for interval u in D.

Figure 3.9. A configuration and a subgraph of the access graph illustrating a node of degree 3.

Since the number of arcs in a graph is equal to half the total degree in a graph [27], the number of arcs in G can be no greater than $\frac{3}{2}\eta$ where η is the number of nodes in G. The theorem that follows gives a more exact bound

on the number of arcs in G. It shows that the number of arcs in G is of complexity $\mathcal{O}(c,n)$ where c is the number of segments in D.

Theorem 3.1: Let n be the number of points in a configuration D, and c the number of segments. Then the number of arcs k in the unconstrained access graph G for D is

$$k = 3n-c-3 .$$

Proof: Let u be the number of uncovered points in D, v be the number of single-covered points, and w the number of double-covered points. Then $n = u+v+w$. Each uncovered point p_i implies two arcs in G, namely, the arcs corresponding to feasible paths from interval u_{i-1} to u_i in both streets S^+ and S^- except for point p_i which has no interval to its left. Thus, there are twice as many arcs contributed to G as there are uncovered points in D *minus* two if one of the uncovered points is point p_1.

If we define

$$\lambda = \begin{cases} 0 \text{ if point } p_1 \text{ is uncovered} \\ 1 \text{ if point } p_1 \text{ is single-covered} \\ 2 \text{ if point } p_1 \text{ is double-covered} \end{cases}$$

then the uncovered points in D contribute $2u-(\lambda-1)(\lambda-2)$ arcs to G.

Each routed segment contributes one arc to G unless it is attached to point p_1. Thus, there are c arcs contributed to G by the segments minus one if p_1 is single-covered or two if p_1 is double-covered.

Each single-covered point p_i contributes one arc to G, namely, the arc representing the feasible c-segment from interval u_{i-1} to u_i in the street opposite the one in which the routed segment attached to p_i resides. The one exception to this is when the single-covered point is p_1 in which no arc is contributed. Mathematically, single-covered points in D contribute v arcs minus one arc if point p_1 is single-covered, or $v+\lambda(\lambda-2)$ arcs. In the example of Figure 3.9, feasible c-segments b, c, d, and f create arcs in G as contributed by single-covered points p_2, p_3, p_4, and p_6 respectively.

Finally, noting that double-covered points contribute no arcs to G, and also noting that there are n-1 switching arcs in G, we have

$$k = 2u-(\lambda-1)(\lambda-2)+v+\lambda(\lambda-2)+(c-\lambda)+(n-1)$$
$$= n+2u+v+c-3 .$$

Since $u = c-v-w$, and since each segment has two single-covered points unless two segments have the

same end point, in which case the total number of single-covered points contributed by the segments is reduced by two, then $v = 2n-2w$, and $u = c-2n+2$. Substituting for u and v in the expression for k above, we obtain

$$k = 3n-c-3 .$$ □

Note that the number of arcs in G is a linear function of the number of points n in D and the number of segments c routed in D. By using Theorem 3.1, we can establish the upper and lower bounds on k as shown in Corollary 3.1.

Corollary 3.1: The bounds on the number of arcs in the unconstrained access graph G for a routing configuration D, where D has n configuration points and m original points (i.e., no pseudo-points), are

$$2(n-1) \geq k \geq 2(n-2)+m .$$

Proof: The minimum number of segments c for a given value of n maximizes k. There are $r = n-m$ pseudo-points, each of which, by definition, are double-covered. Thus, there must be at least $4-1$ segments defined on r pseudo-points. The must also be at least two additional segments each with one end point defined on an original point. Thus, $c \geq r+1$. Substituting

this lower bound for c in the expression in Theorem 3.1 results in

$$k \leq 2(n-2)+m \qquad (n \geq m).$$

A maximum of n-1 segments can be defined on n points. Thus,

$$k \geq 2(n-1).$$ □

3.3 Equivalence of a Configuration and its Access Graph

Up to this point, we have discussed the generation of the unconstrained access graph and its size. We now give some theoretical results which show the equivalence of a configuration and its unconstrained access graph. In addition, the proof of the major theorem in this section also provides a method by which a configuration can be generated given an access graph. Even though this method is not furthere exploited in this study, results presented in later chapters take advantage of the fact that a configuration can be obtained from an access graph. The results in this section are applicable (with appropriate extensions) to all access graphs discussed in this dissertation.

To prove equivalence, it is first necessary to show that every configuration has an access graph, and, furthermore, that this graph is unique.

Lemma 3.1: A configuration defines a unique access graph.

Proof: Existence is proven by definition of an access graph. There is only one access graph for a configuration because (1) each channel in the configuration is represented by a path containing at least one arc in the access graph, (2) no more than one path is defined by a channel, and (3) each street arc in the access graph represents one and only one distinct *canonical* channel in the configuration. Thus, there is a 1-1 mapping of canonical channels in a configuration to arcs in the access graph. Furthermore, the mapping from intervals (which contain the configuration points) is also a 1-1 mapping to the triplet (V^-,V^+,W) where V^- is a node representing the lower street side of an interval, V^+ is a node representing the upper street side of an interval, and W represents the switching arc between V^- and V^+. Note that although W always exists in the unconstrained graph, it may not exist in the constrained access graphs presented in the next two chapters. Nodes V^- and V^+ exist for every interval in the configuration by definition. Therefore, the access graph for a configuration always exists and is unique. □

Before presenting the next theorem, we present the concept of a dual configuration and give a simple procedure for obtaining it and the access graph (called a *dual access graph*) for the dual configuration.

A *dual configuration* is a configuration, as defined in Chapter 2, with the additional characteristics:

1. tree connections, as well as chain connections, are permitted
2. more than one segment may be drawn between the same two points, and
3. the dual configuration has $m = \frac{n}{2} + 1$ points where the access graph from which the configuration is obtained has no nodes.

Figure 3.10 shows an example of a dual configuration where m = 11.

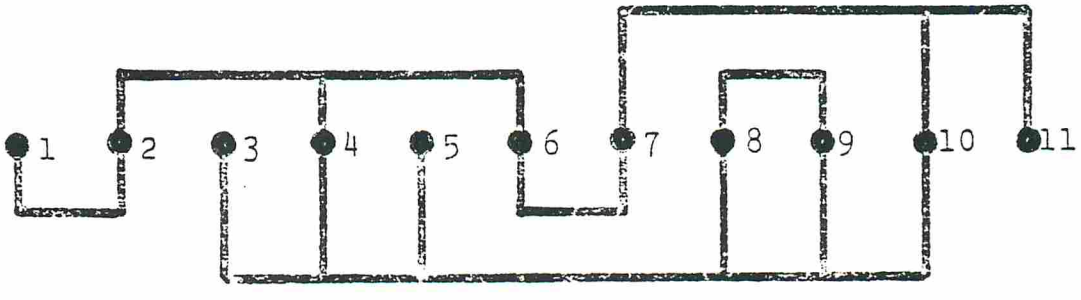


Figure 3.10. A dual configuration D^*.

A dual configuration D^* is obtained from an access graph G by the following simple procedure.

1. For every pair of nodes (i^+,i^-) in G, draw a point i on the point line to the right of point (i-1).

2. For every street arc (x^+,y^+) or (x^-,y^-) in G, draw a rectilinear path between points x and y such that it does not intersect any paths already drawn. Note that tree connections may occur.

3. Add a point to the left of point 1 on the point line. Label it 0.

4. Relabel all points such that i = i+1.

5. If point 1 is uncovered in street S^X of D draw a rectilinear path between points 1 and 2 in street S^X of D^*. This completes the construction of the dual configuration D^*. □

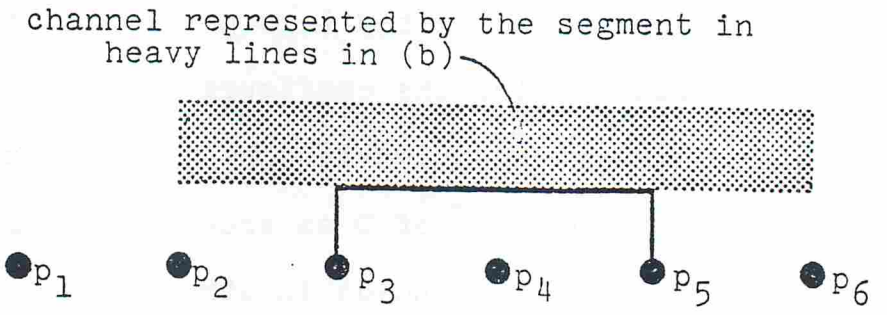
The reason one can draw non-intersecting paths in D^* in step 2 is because street arcs in G do not intersect. (In fact, G is planar since the configuration D from which it is obtained is planar, by definition, and the arcs in G represent non-intersecting channels in D.) Furthermore, the points in D^* lie along a line in the same sequence as the points in D.

In the proof of the equivalence between D and G it will

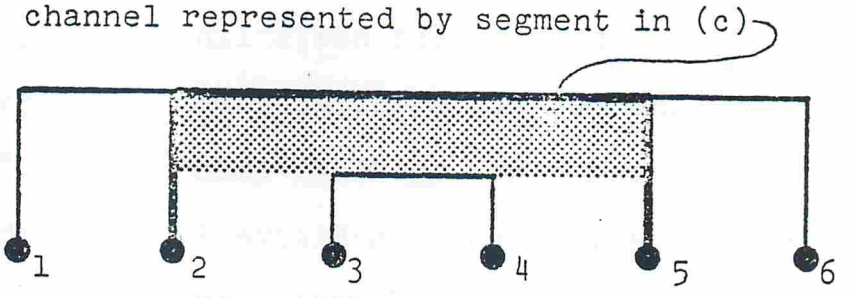
be necessary to construct a configuration D′ from D^*. The relabeling done in step 4 is performed so that the channels of D^* will define the configuration D′ which possesses the same segment end-point labels as D. Consider a segment (u,v) in street S^+ of D as shown in Figure 3.11a. (This same argument also holds in street S^-). The outer canonical channel of this segment is between intervals (u-1) and v. If the relabeling is not performed in constructing D^*, then the channel in D is represented by the segment (u-1,v) in D^* as shown by the heavy lines in Figure 3.11b. If we then construct the configuration D′ where the segments in D′ represent the canonical channels in D^*, we obtain the configuration shown in Figure 3.11c which contains only the segment (u-1,v-1). However, by relabeling each point in D^* to be i = i+1 as in step 4 of the construction procedure, we obtain the segment (u,v) in D′. We take advantage of this feature to prove that D′ = D in Theorem 3.2.

The reason a path between points 1 and 2 in D^* is drawn if point 1 is uncovered in D is to ensure that a channel does not exist in D^* with one end at interval 1. If such a channel were to exist in D^* then D′ would contain a segment with point 1 as one of its end points. Since we desire D′ to equal D, such a segment is undesired in D′ if in fact there is no segment attached to point 1 in D.

As an example of the generation of a dual configuration, consider the configuration D in Figure 3.12. The



(a) Configuration D ($u = 3$, $v = 5$).



(b) Configuration D^* without point relabeling ($u = 3, v = 5$).

(c) Configuration D' resulting from D^* shown in (b).

Figure 3.11. A sequence of configurations (upper street only) demonstrating the labeling of points.

access graph G for D is shown in Figure 3.13.

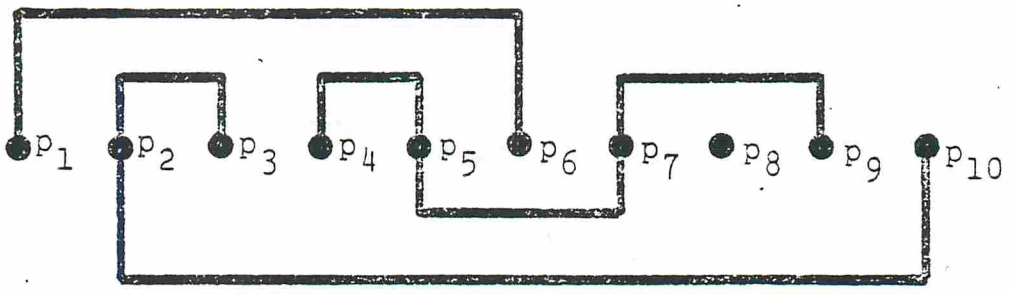


Figure 3.12.. A configuration D.

The dual configuration D^* for D as obtained from graph G using the dual configuration construction procedure is shown in Figure 3.10. Note that path $(1^-,2^-)$ is constructed as a result of step 5 in the construction procedure.

A comparison of the dual configuration in Figure 3.10 with the configuration in Figure 3.12 shows several interesting features. First, the number of points in D^* is one more than the number of points in D. Second, D^* is planar, has tree as well as chain connections (e.g., connections in both streets at point 4), and has two segments attached to the same points (e.g., the segments in opposite streets attached to points 8 and 9). Finally, with the exception of the channel between intervals 10 and 11 in the lower street of D^*, every channel in D^* corresponds to an arc in D (hence the term "dual"). To further examine this point,

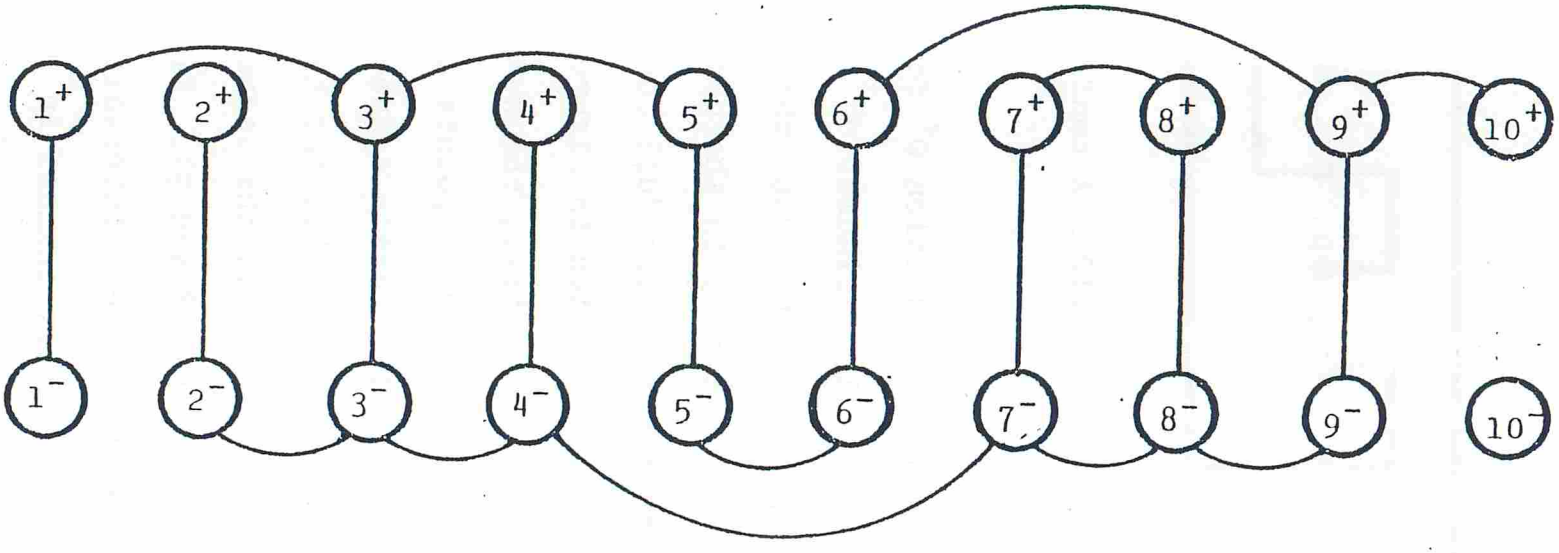


Figure 3.13. The access graph G for D.

consider Figure 3.14 which is the dual access graph G^* for D^*. G^* is defined in the same manner as an access graph with the exception that nodes $(m+1)^+$ and $(m+1)^-$ (as well as adjacent arcs) are not created. Algorithm 3.1 for generating an access graph can be used to generate the dual access graph if the configurations shown in Table 2.2 (and referenced by the jump table in Algorithm 3.1) are augmented with tree connections as shown in Table 3.4.

Table 3.4. Adjacent Point Configurations with Tree Connections

Configuration in Table 2.2	ADD:
4	
5	
7	
8	
9	  

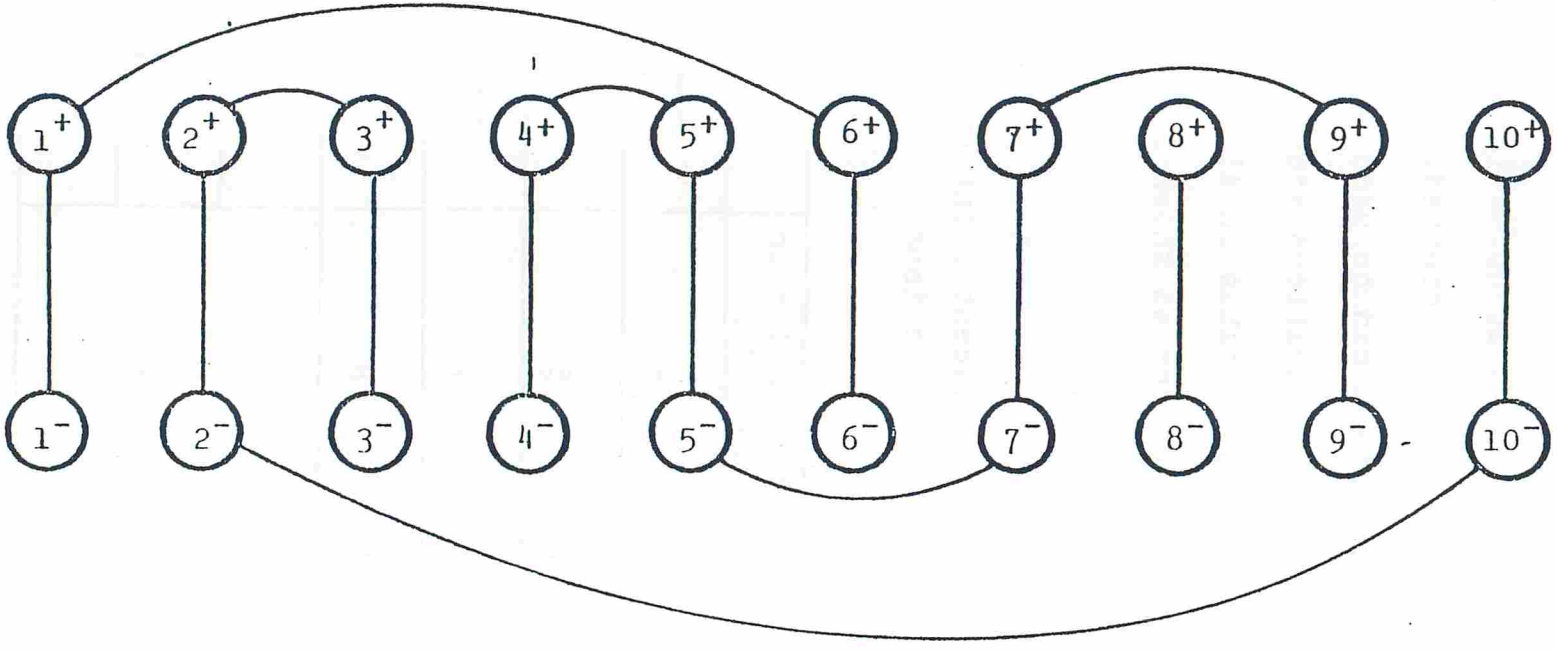


Figure 3.14. A dual access graph G^*.

If we now create a configuration (see Figure 3.15) from the configuration G^* by drawing a point i to the right of point (i-1) on the point line for every pair of nodes (i^+,i^-) in G^*, and for every strict arc (x^+,y^+) (or (x^-,y^-)) in G^* a rectilinear path between points x and y is drawn such that no paths intersect, we obtain a configuration D' which, as can be seen, is the same as D. Thus we see that one can obtain the configuration which generates an access graph when given only the graph.

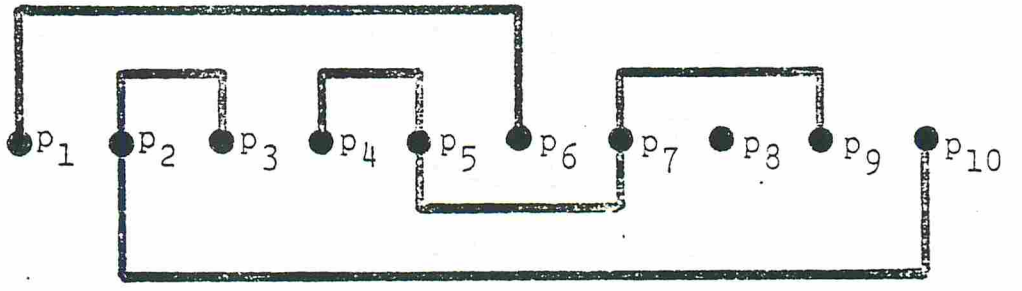


Figure 3.15. The configuration D' = D.

The foregoing discussion is the foundation for Theorem 3.2.

Theorem 3.2: An access graph defines a unique configuration D'. If D is the configuration from which the access graph is obtained, then D' = D.

Proof: Let G be the access graph for a configuration D. Let D' be the configuration resulting from the following sequence of constructions as described above:

$$G \rightarrow D^* \rightarrow G^* \rightarrow D' .$$

First, note that D^* is uniquely derived from G since (1) each node pair in G maps onto a unique point in D^*, and (2) each street arc in G maps onto a unique path in D^*. Second, G^* is uniquely derived from D^* by Lemma 3.1. Finally, D' is uniquely derived from G^* by the same argument above for the G-to-D^* construction. Thus, D' is uniquely obtained from G.

To show that D' = D, let (u,v) be a segment routed between points u and v in street S^+ in D. (This same argument holds for a segment in street S^- as well.) Then the outer canonical channel of segment (u,v) is represented by an arc in G between nodes $(u-1)^+$ and v^+. This arc is represented by the segment (u,(v+1)) in D^* as a result of the construction presented earlier in this section. The

inner channel (which is canonical) of segment $(u,(v+1))$ in D^* is represented by an arc (u^+,v^+) in G^* which is represented by a segment (u,v) in street S^+ of D'. Thus, a segment (u,v) in D is mapped into a segment (u,v) in D'. By extending this argument to every segment and uncovered point in D, and imposing the constraints on point m+1 in D^* (and nodes $(m+1)^+$ and $(m+1)^-$ in G^*) as defined by the dual configuration and dual access graph construction procedures stated previously in this section, the equivalence of D' in D is shown. □

The equivalence between a configuration and its access graph may now be proven.

Theorem 3.3: A configuration and its access graph are equivalent, i.e., they map 1-1.

Proof: By Lemma 3.1, the access graph for a configuration exists and is unique. Likewise, by Theorem 3.2, the configuration obtained from an access graph exists and is unique. Furthermore, by the same theorem, the configuration obtained from an access graph is identical to the configuration which generates the access graph. Thus, equivalence between a configuration and its access graph is shown. □

3.4 Finding a Minimum-Length Path in a Unidirectional Configuration

With the unconstrained access graph G now defined, it is an easy matter to find a minimum length path for a net N = (s,t) between points s and t in a configuration. It is important to note that the steps outlined here for finding a minimum-length path for a net in an *unconstrained* configuration are identical to those required to find minimum-length feasible paths in both fixed-track and floating-track constrained configurations. The constrained cases are explored in the next two chapters.

Consider a net N = (s,t) with end points s and t where s and t are two points in a unidirectional configuration D. Let G be the unconstrained access graph for D. Then, with appropriate consideration of the first and last points, a least-cost path is found in G between these two points. Since point s (or t) in D is represented by two nodes s^+ and s^- (or t^+ and t^-) in G, we must select the first and last path points from s^+, s^-, t^+, and t^- such that the selected nodes are uncovered, and there are no shorter paths between any nodes s^+, s^- and t^+ or t^-. For example, consider the configuration shown in Figure 3.16. The unconstrained access graph for D is shown in Figure 3.17.

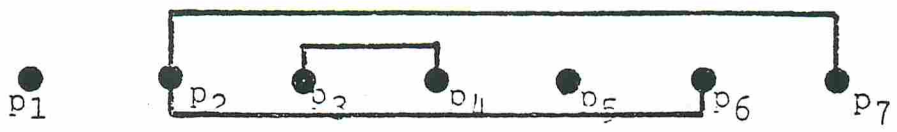


Figure 3.16. A configuration D which illustrates end-point selection for determining minimum-length paths.

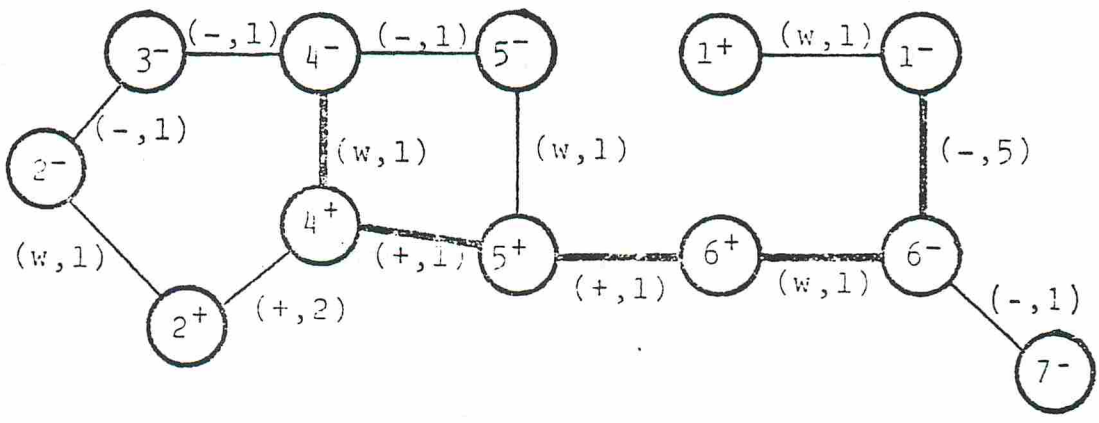


Figure 3.17. The unconstrained access graph G for the configuration shown in Fig.3.16.

Assume we wish to route net $N = (p_1, p_4)$. By noticing that point p_4 in D is covered in street S^+, we note that one end of the path to be found in G may *not* be node 4^+. Therefore, it must be node 4^-. Also, noticing that point p_1 is completely uncovered in D, we are free to choose either node 1^+ or 1^- in G as the other end of the path to be found in G. However, if we choose node 1^+ to be the end node, any path we find between nodes 4^- and 1^+ will be greater in length by 1 unit than any path between nodes 4^- and 1^-. Thus, we choose node 1^- to be the other end of the path. The heavy arcs represent a minimum-length path between nodes 1^- and 4^-.

The approach taken here to find the appropriate end nodes is: if point s is uncovered in D then "collapse" nodes s^+ and s^- to a single node s (similarly for nodes t^+ and t^-) by forcing switching arcs (s^+,s^-) $((t^+,t^-))$ to a length of zero. After the least-cost algorithm is run between s and t in G, we select the appropriate end nodes for the path.

If s (or t) is covered in, say, street S^+, then choose nose s^- (or t^-) as the end node. No node collapsing is necessary in this case.

3.4.1 Algorithm 3.2. Minimum-length path routing in an unconstrained configuration

Purpose: This algorithm determines a feasible path P for a net N in an unconstrained configuration D such that P

is of minimal length.

Method: This simple procedure operates such that the end-nodes in G are first determined, and then the least-cost algorithm as presented in Chapter 2 is executed. The sequence of nodes thus found exactly represents a minimal-length path for net N in D.

Input: A net N = (s,t) and an unconstrained access graph G representing a unidirectional configuration D.

Output: A minimal-length path for N in D.

Procedure:

Step 1. Select the path end points in G as follows:

(a) if both s^+ and s^- are covered, no solution exists.

(b) if neither s^+ nor s^- is covered, collapse them to a single node by forcing arc (s^+,s^-) to have a length of zero.

(c) if s^+ (s^-) is covered and s^- (s^+) is not, choose s^- (s^+) as the end point.

(d) perform steps a-c above replacing s by t.

Step 2. Find a least-cost path in G between the end points found in step 1 using the least-cost algorithm presented in Chpater 2, Section 2.11. The cost of each arc is the length of the segment or switch represented by the arc.

Step 3. If either (or both) of the end points were obtained by collapsing nodes in step 1, then determine the appropriate end points as follows. Let the path found in step 2 be defined by node sequence s, $z_1, z_2, \ldots, z_{b-1}, z_b$, t and let the label on z_1 be x (i.e., x = + or -). Then s^x is the appropriate end node adjacent to z_1. Similarly, t^y is the appropriate end node adjacent to z_b which has label y. □

In this procedure, the path found in G represents the path in D for N.

3.4.2 Timing analysis of Algorithm 3.2

Steps 1 and 3 require essentially constant time. Thus, step 2 defines the complexity for Algorithm 3.2. As shown in Chapter 2, a least-cost path may be found in G with no more than $6n+(5n+1)\log_2(2n-1)+\log_2(n)$ operations. Algorithm 3.2 is therefore of complexity $\mathcal{O}(n\log n)$.

3.4.3 An example

Consider the routed configuration D in Figure 3.12, and assume we wish to find the minimum length path for net N = (p_3, p_5). First, the access graph representing D is obtained using Algorithm 3.1. The resulting graph is shown in Figure 3.13. Each point in D is assumed to be separated by two units of length. The minimum-length path

procedure (Algorithm 3.2) is executed next. The heavy arcs reflect the path that was found for this example.

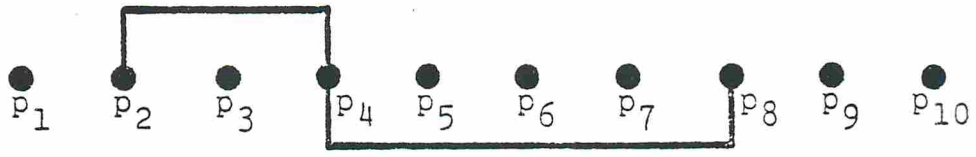


Figure 3.18. A configuration illustrating the minimum-length routing of net $N = (p_3, p_5)$.

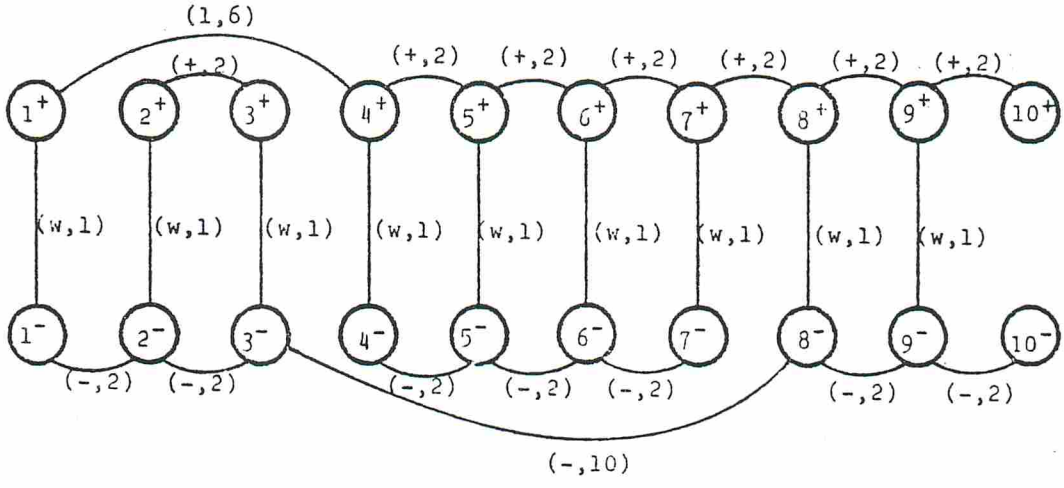


Figure 3.19. The access graph for the configuration shown in Fig. 3.18.

Finally, the configuration D is updated to a new configuration which contains the path for N. This configuration is shown in Figure 3.20. Note that a pseudo-point was added between points p_1 and p_2.

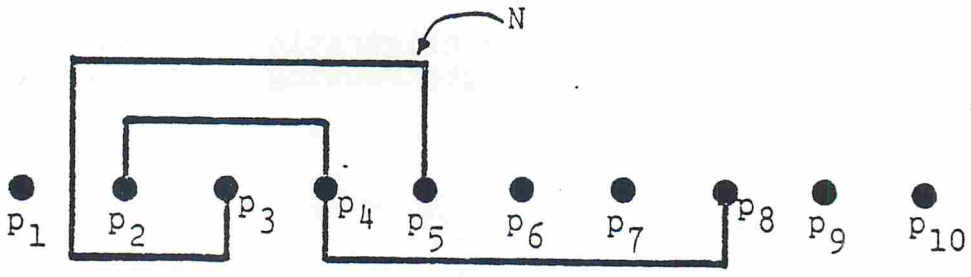


Figure 3.20. The final configuration containing the minimum-length routed path for net N.

3.5 Conclusions

Given that one is not concerned about the restrictions imposed by tracks when attempting to route nets in a unidirectional environment, the minimum-length routing can be efficiently performed given the results presented in this chapter. The method for routing presented here can be used for practical problems. Using a simple track assignment algorithm, for instance, scan the points p_i from left to right incrementing a counter CNTR+ (CNTR-) if point p_i

is covered in street S^+ (S^-) *and* p_i is the left-most point of the attached segment. (CNTR+ (CNTR-) is the track assigned to the segment attached to p_i. Decrement CNTR+ (CNTR-) if p_i is the right-most point of the attached segment. No guarantee is given that routes will be within street track capacities. However, if one or more paths exist for a route, the results given here guarantee that a path will be found.

If we assume that a large number of nets are to be routed in an unconstrained configuration with n points, then the total routing process requires $\mathcal{O}(n^2 \log n)$ operations. Ting and So's method requires $\mathcal{O}(n^2)$ operations. Thus, the two methods are similar in efficiency. The results given in this chapter are noteworthy for their capability of finding the optimal solution.

CHAPTER 4

FIXED-TRACK UNIDIRECTIONAL ROUTING

4.0 Introduction

In this chapter a somewhat more complicated unidirectional structure is considered than that studied in the previous chapter. Fortunately, the basic model (i.e., the unconstrained access graph) is easily extended to account for the constraints introduced. We also introduce and study the notion of *congestion* with respect to the density of routes in a unidirectional configuration. Given the assertion that minimizing congestion leads to a more routable board, we show how to use an appropriately labeled access graph to route a net such that congestion is minimized.

4.1 The Fixed-Track Unidirectional Configuration

Whereas in the previous chapter no constraints were placed on tracks (in fact, we ignored them altogether), here we assume there is a limit to the number of both street and switching tracks. Also, we assume that once a feasible section has been assigned to a track for routing purposes, it cannot be moved to a different track to accommodate future paths — hence, the name "fixed track" routing. Some definitions relating to fixed-track configurations are now presented.

4.1.1 Street capacity

Let D be a unidirectional configuration and consider x to represent either + or -. The number of tracks in street S^x is defined to be the *street capacity* T^x of street S^x, and the number of tracks in each interval u_i in D is called the *interval capacity* $|u_i|$.

4.1.2 Section length

In Section 2.5 of Chapter 2, the notion of length was presented as a general concept and the units of length were not specified. With the fixed-track case (as well as the floating-track case dealt with in Chapter 5) we are in a position to specify the units of length.

The location α_i of point p_i in D is the total number of vertical tracks (occupied or unoccupied) to the left of and included in p_i. For example, the location of each point is shown in the configuration in Figure 4.1.

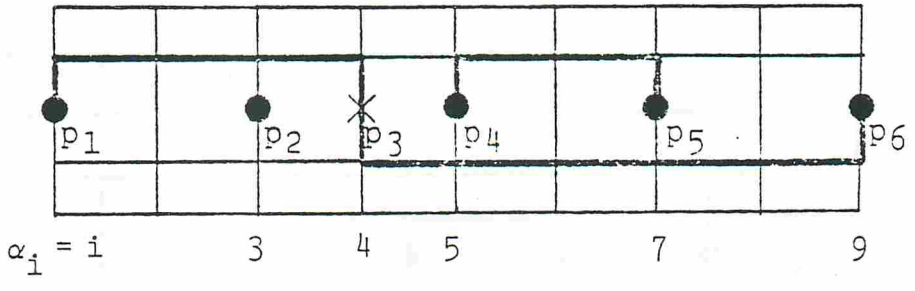


Figure 4.1. An illustration of the location of each point in a configuration.

It is obvious that since the length of a section (p_i,p_j) is $|\alpha_j-\alpha_i|$ the length of a section is the total number of switching tracks contained within the span of the section.

4.1.3 Inner and outer channels

As defined in Chapter 2, a channel is an area of the unidirectional configuration through which a wire can be routed (if the channel contains unused tracks).

A section channel is further defined to be an *inner channel* with respect to a section γ if it is to the inside of γ and an *outer channel* if it is to the outside of γ. If γ is in track t_i and the nearest adjacent inner section is in track t_j, then the inner *channel capacity* ρ of the channel bounded by tracks t_i and t_j is t_i-t_j-1. If the nearest adjacent outer section to γ is in track t_k then the outer channel capacity ρ is t_k-t_i-1. Figure 4.2 illustrates an inner and outer channel.

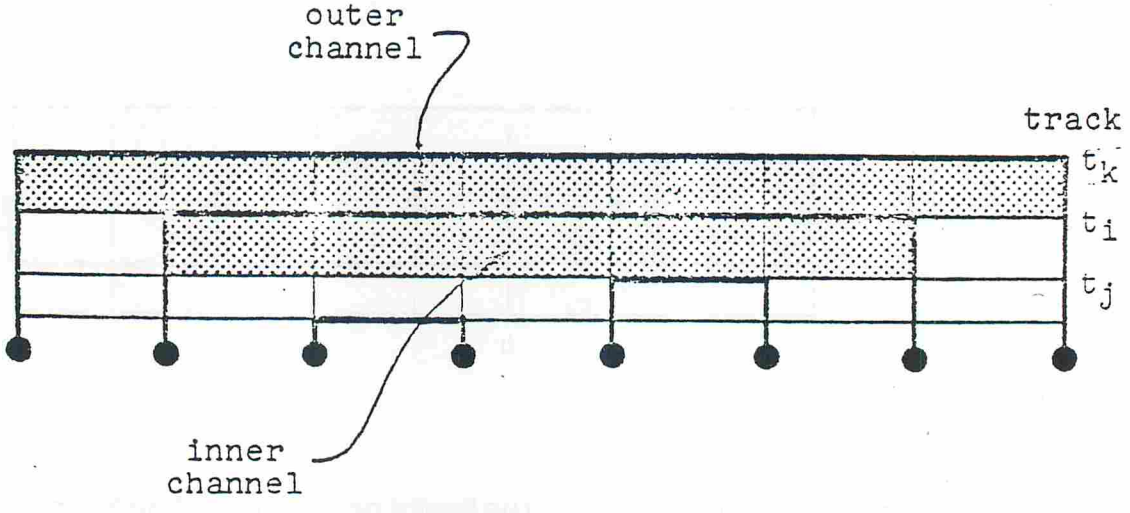


Figure 4.2. Configuration illustrating the concept of inner and outer channels.

4.2 The Access Graph for the Fixed-Track Routing Problem

The fixed-track access graph is, by definition, a subgraph of the access graph defined in the previous chapter for the unconstrained case. In addition, each arc is labeled with:

1) the type of arc (i.e., "+," "-," or "w"),

2) the length, δ, of the channel in the configuration represented by the arc,

3) the channel capacity, ρ, of the channel represented by the arc. The reason for determining ρ will become obvious in Sections 5 and 6 dealing with minimum congestion paths,

and

4) the track, t, occupied by the routed section bounding the inner side of the channel represented by the arc.

At first glance, it would appear that a way to produce the fixed-track access graph for a configuration would be to create the unconstrained access graph, label arcs appropriately, and then remove arcs (and possibly nodes) where necessary. Although this approach is feasible, a more straightforward method is presented here. The algorithm given next is derived from Algorithm 3.1.

Algorithm 4.1: Construction of the fixed-track access graph.

Purpose: This algorithm generates a weighted unidirected graph G, called an access graph for a given routed configuration D where street capacities are considered, and track assignments for routed paths remain unchanged.

Method: The point line is scanned twice from left to right, once for street S^+ and a second time for street S^-. During each scan successive adjacent points p_i and p_{i+1} are examined. Appropriate action is taken in steps 5-11 of the procedure based upon which one of nine adjacent point configurations (see Table 2.2, Chapter 2) is applicable. Steps 5-10 are performed only if tracks are available for the feasible segment being processed. Stack and graph operations are performed such that at the conclusion of the second point scan, the entire graph is constructed. The algorithm uses a stack S which contains elements of the form (j,μ_j), where j represents a point, and μ_j is the inner track adjacent to the section bounding the outer channel of the feasible segment whose left point is p_j. We define point p_{n+1} to be an uncovered imaginary point lying just to the right of point p_n on the point line in D.

Input: Routed configuration D including the street tracks assigned to each segment, the number $|u_i|$ of free switching

tracks in each interval u_i, and the number α_i of switching tracks to the left of each point p_i.

Output: The fixed-track access graph for D.

Procedure:

Step 1. (initialization) Set x = "+" and set stack S empty. Construct the 2n nodes $1^+,2^+,\ldots,n^+$, $1^-,2^-,\ldots,n^-$.

Step 2. (switching arcs) For i = 1,2,...,n-1, make an arc between nodes i^+ and i^- if $|u_i| > 0$. Label each arc thus created as $(w,1,|u_i|,\alpha_i)$.

Step 3. Set i = 0. Push item $(0,T^x)$ on stack S.

Step 4. (begin main loop) Set i = i+1. Let the top item of stack S be (j,μ_j). Consider points p_i and p_{i+1} in D. Select the applicable case from Table 4.1 and branch to the step indicated. t_i is the track occupied by the segment attached to point p_i in D. If there is no such segment, then $t_i = 0$.

Step 5. Pop stack S to get j and μ_j. If $j \neq 0$ and $t_i < \mu_j$, construct arc (i^x,j^x) and label it $(x,\alpha_i-\alpha_j,\mu_j-t_i,t_i)$. Push (i,μ_j) onto stack S regardless of the value of j or t_i. Go to step 11.

Step 6. Push (i,t_i-1) onto stack S. Go to step 11.

Step 7. Pop stack S to get j and μ_j. If $\mu_j \neq 0$, construct arc (i^x,j^x) and label it $(x,\alpha_i-\alpha_j,\mu_j,0)$. Go to step 11.

Step 8. Pop stack S to get j and μ_j. If $\mu_j \neq 0$ and $j \neq 0$, construct arc (i^x,j^x) and label it $(x,\alpha_i-\alpha_j,\mu_j,0)$. Push (i,μ_j) onto stack S regardless of the value of μ_j or j. Go to step 11.

Step 9. Pop stack S to get j and μ_j. If $t_i < \mu_j$, construct an arc (i^x,j^x) and label it $(x,\alpha_i-\alpha_j, \mu_j+t_i,t_i)$. Go to step 11.

Step 10. (end of main loop) If $i \neq n$, then go to step 4.

Step 11. If x = "+," then set x = "-," set stack S empty, and go to step 3. Otherwise, exit procedure. □

This algorithm is similar to the access graph construction algorithm in Chapter 3 with the exceptions

1) that arcs are labeled with 4-tuples rather than 2-tuples, and
2) arcs are not created if the channels represented by the arcs contain no free tracks.

Table 4.1 Jump Table for Algorithm 4.1.

Adjacent Point Configuration (Table 2.2,Ch.2)	p_i is covered in S^x	p_{i+1} is covered in S^x	p_i is left point of segment	p_{i+1} is left point of segment	Go to Step
1	no	no	—	—	5
2	yes	no	no	—	5
3	no	yes	—	yes	8
4	no	yes	—	no	7
5	yes	yes	no	no	9
6	yes	yes	no	yes	5
7	yes	no	yes	—	6
8	yes	yes	yes	yes	6
9	yes	yes	yes	no	10

4.2.1 Example of operation of Algorithm 4.1

To gain insight into the operation and use of Algorithm 4.1, consider the following example. A routed configuration consisting of four nets is routed as shown in Figure 4.3.

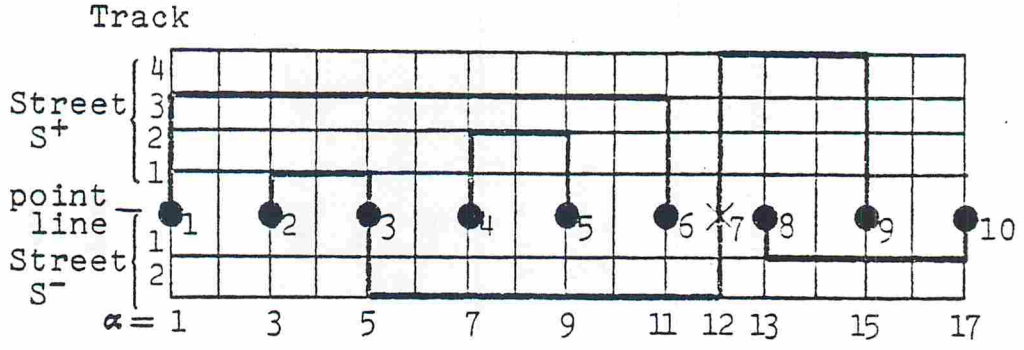


Figure 4.3. Routed configuration used as an example of the operation of Algorithm 4.1.

We first construct and label the twenty nodes of G. Arcs are then inserted (and labeled) between all node pairs i^+ and i^- except for nodes 6^+, 6^-; 7^+, 7^-; 10^+ and 10^-

since there are no available vertical tracks in intervals 6, 7, and 10.

Starting with i = 1 in step 4, and with stack S containing one element, (0,4), we branch to step 6. The track assigned to section (1,6) is 3, so we push (1,2) onto stack S (Figure 4.4a). Next time step 4 is processed, i = 2, and control branches to step 10 where no action on stack S or graph G is taken. When step 4 is again entered, i = 3, and control branches to step 5. The track assigned to section (2,3) is 1 which is less than $\mu_j = 2$. $j \neq 0$, so arc $(1^+,3^+)$ is created and labeled (+,4,1,1). (3,2) is pushed onto stack S since i = 3, and control branches to step 10 (Figure 4.4b). The next time thorugh step 4, control branches to step 10 where no action on stack S or graph G is taken. Step 4 is again entered with i = 5 where, branching to step 9, we see $t_5 = 2$ which is not less than $\mu_j = 2$. Thus, no arc is created. Figure 4.4c shows the stack as it exists at this point. Continuing in the same manner to complete the scan of points for x = "+" we obtain the results shown in Table 4.2. Completing the processing of Algorithm 4.1 by performing the scan of points with x = "-" results in the access graph shown in Figure 4.5 with solid lines for arcs. The dotted arcs represent those arcs that would be in G if D were unconstrained.

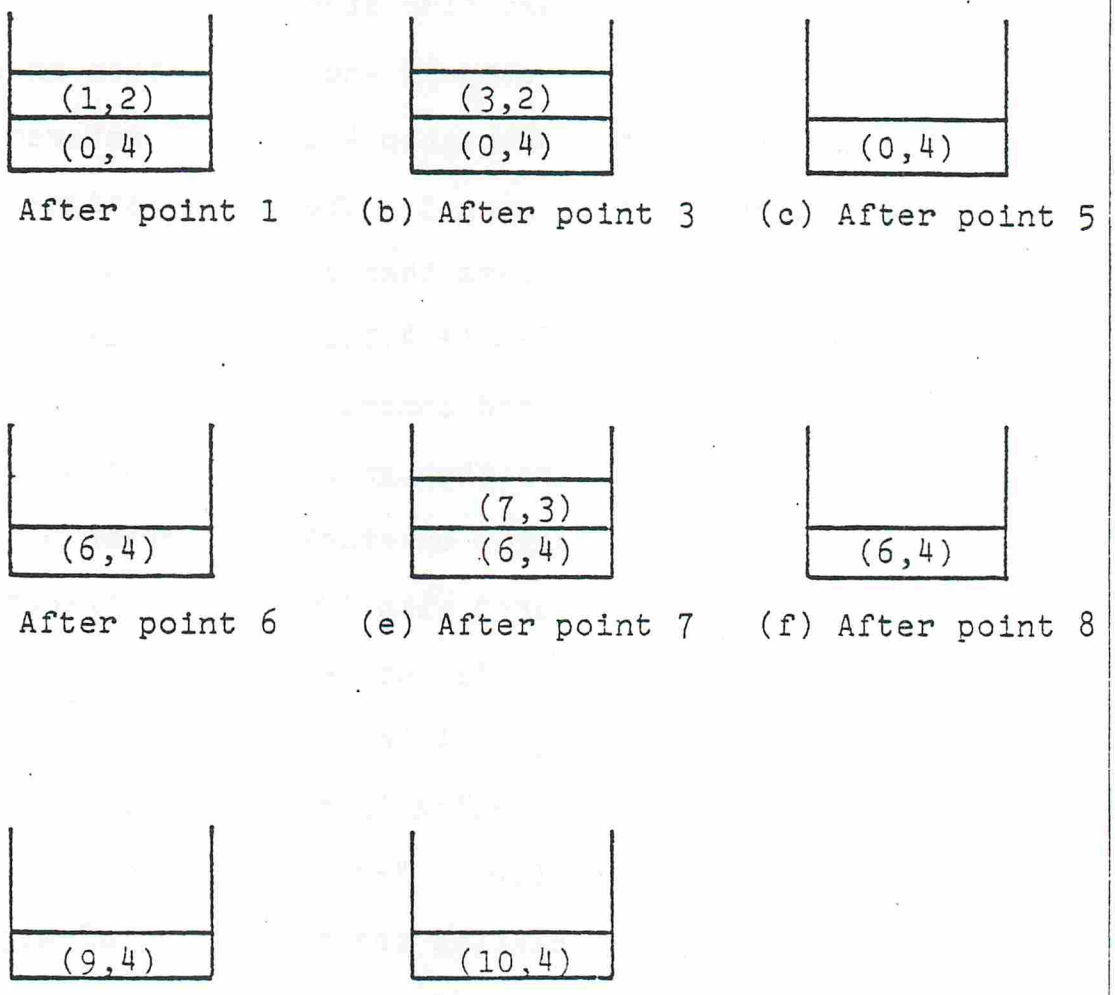


Figure 4.4. Stack S configuration at various points of execution of Algorthm 4.1 with the configuration shown in Figure 4.3 as input.

Table 4.2. Processing Steps for an Example of the Processing of Algorithm 4.1.

i	Process Step	Stack Action	Stack Fig.4.4	Arc Created	Label of Arc
1	6	Push (1,2)	(a)	–	–
2	11	–	–	–	–
3	5	Pop (1,2) Push (3,2)	(b)	$(1^+,3^+)$	(+,4,1,1)
4	10	–	–	–	–
5	9	Pop (3,2)	(c)	–	–
6	5	Pop (0,4) Push (6,4)	(d)	–	–
7	6	Push (7,3)	(e)	–	–
8	7	Pop (7,3)	(f)	$(7^+,8^+)$	(+,1,3,0)
9	5	Pop (6,4) Push (9,4)	(g)	–	–
10	5	Pop (9,4) Push (10,4)	(h)	$(9^+,10^+)$	(+,2,4,0)

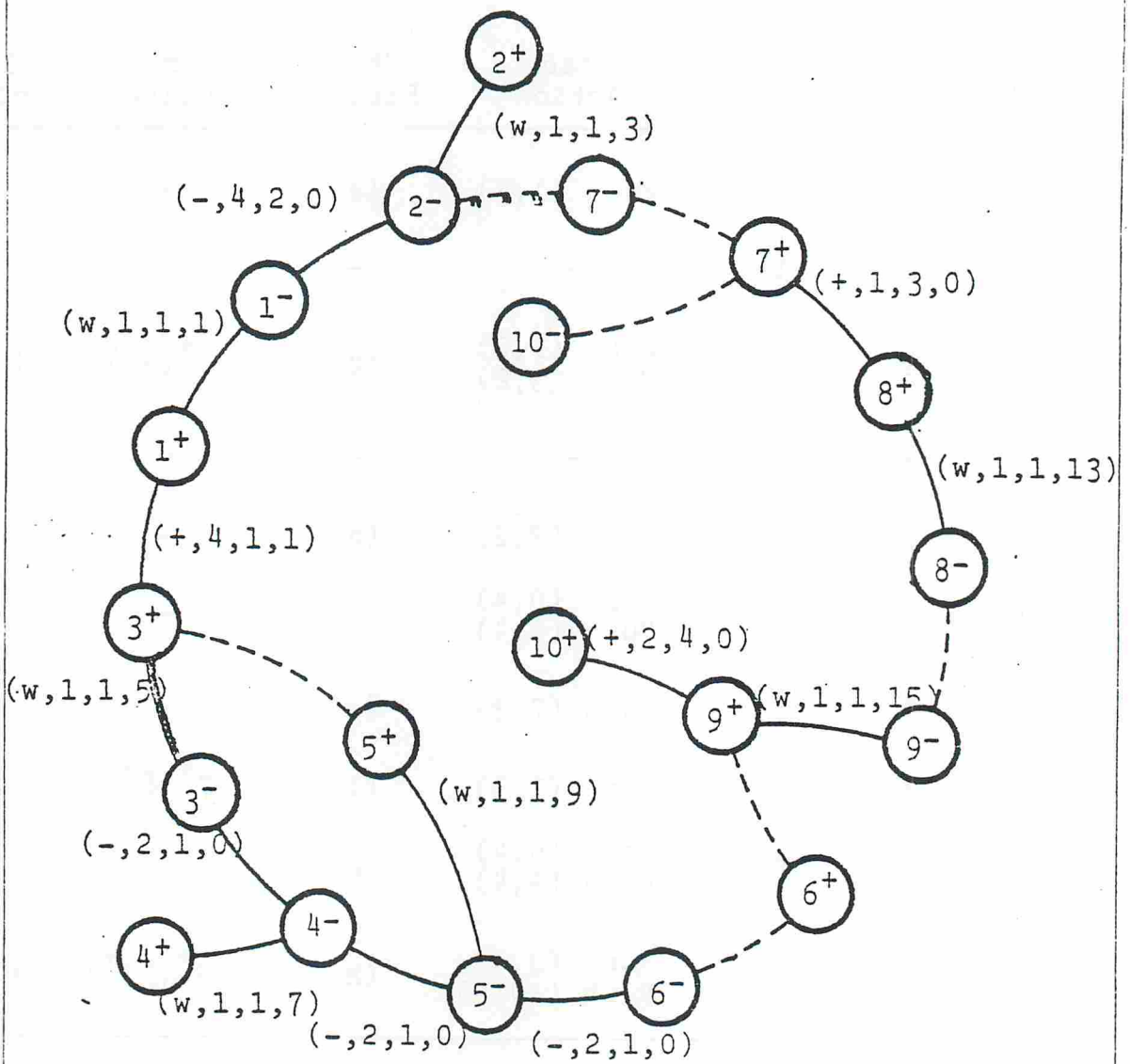


Figure 4.5. Access graph for routed configuration D shown in Figure 4.3. Dotted arcs are present if tracks are ignored (i.e., unconstrained access graph).

4.2.2 Execution time of Algorithm 4.1

The derivation of the execution times for steps 5-10 of the algorithm is based on the same execution time analysis for Algorithm 3.1 in the previous chapter. Fortunately, as we show below, Algorithm 4.1 is sufficiently similar to Algorithm 3.1 that we can use the configuration requiring maximum and minimum running times for Algorithm 3.1 to determine the maximum and minimum running times for Algorithm 4.1.

Consider a configuration D with n points. Then the first three steps of Algorithm 3.1 differ from the first three steps of Algorithm 4.1 only by the inclusion of a test for switching channels in step 2 of Algorithm 4.1. The last two steps of each algorithm are identical. Thus, referring to Table 3.2, steps 1, 2, 3, 11, and 12 of Algorithm 4.1 require

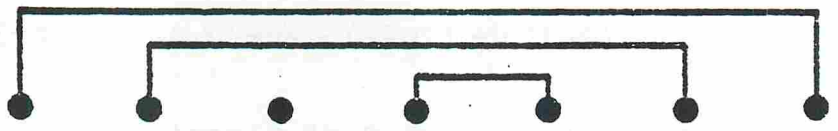
$$(2n+2) + (2n-2) + 2n+4 + n-1 = 7n+3 \text{ operations .}$$

Steps 4 through 9 of Algorithm 4.1 can be mapped into similar steps of Algorithm 3.1 as shown:

Algorithm 3.1	Algorithm 4.1	Number of Operations Each Time Entered
step 4	step 4	2
step 5	steps 5,8	4
step 6	step 6	2
step 7	steps 7,9	4

Because of the mapping shown above, we can use the same adjacent point configuration analysis as carried out in the previous chapter to derive a configuration requiring the maximum number of operations by Algorithm 4.1 as well as a configuration requiring the minimum number of operations. These configurations are shown in Figure 3.8 and repeated here in Figure 4.6.

(a)



(b)

Figure 4.6. Examples of unidirectional configurations requiring a maximum and minimum number of operations per interval for steps 4 - 10 of Algorithm 4.1.
(a) The configuration requiring a maximum number of operations.
(b) The configuration requiring a minimum number of operations.

For the maximum case, step 5 is repeatedly entered n-1 times for each scan of the points. Thus, 4(n-1) operations are performed by Algorithm 4.1 excluding steps 1, 2, 3, 10, and 11 for one scan of the configuration points. If we include these steps, Algorithm 4.1 requires, in the worst case,

$$\mathcal{O}ps_{max} = 7n+3+8(n-1) = 15n-5 \text{ operations}.$$

For the minimum case, we again refer to Section 2.2.1 of the previous chapter where we note that either of the node sequences

$$B^k(AD)^mC^k \quad \text{or} \quad B^k(DA)^mC^k, \qquad k \geq 0,\ m \geq 0$$

for the simplified precedence graph shown in Figure 3.7 applies. The number of operations reflected by these nodes for Algorithm 4.1 is:

node	operations
A	4
B	2
C	4
D	0

Thus, assuming a track always exists if needed, the total number of operations corresponding to the simplified precedence graph node sequences for the minimum configura-

tion case (for one scan of the points) is

$$2k + 4m + 4k = 6k + 4m .$$

The number of intervals represented by the node sequences is

$$k + 2m + k = 2(m+k) .$$

Therefore, the minimum number of operations required by steps 4-10 of Algorithm 4.1 for one scan of the configuration point is

$$\min_{m,k}\left[\frac{6k+4m}{2(m+k)}\,(n-1)\right] = \min_{m,k}\left[(n-1)\left\{2+\frac{k}{m+k}\right\}\right]$$

$$= 2(n-1) \quad \text{where} \quad k = 0 .$$

It is interesting to note that this is the same result as the minimum number of operations for steps 4-9 of Algorithm 3.1. Finally, we see that the minimum number of operations ∂ps_{min} for the entire Algorithm 4.1 is

$$\partial ps_{min} = 7n+3+4(n-1) = 11n-1 .$$

4.2.3 Bounds on the size of the fixed-track access graph

The same bounds on the size of the unconstrained access graph as developed in Section 3.1.4.2 are applicable to the size of the fixed-track access graph since the fixed-

track access graph is a subgraph of the unconstrained access graph. However, we can derive an upper bound for the number of arcs k in the fixed-track access graph in terms of the number of original points m in the configuration, since n can be no greater than $m(1+|u|)$ where $|u|$ is the number of switching tracks in each original interval.

First, we need to show that a fixed-track access graph for a configuration is a subgraph of the unconstrained access graph for the same configuration. We define a configuration D_1 to be *route-equivalent* to a configuration D_2 if 1) D_1 has the same number of points as D_2, and 2) for every segment (a,b) in S^+ of D_1 there is a segment (a,b) in S^+ of D_2.

In addition, we define a graph G_1 to be a *topological subgraph* of a graph G_2 if G_1 is a subgraph of G_2 ignoring all arc labels.

Lemma 4.1: Let G_u be an access graph for an unconstrained configuration D_u, and let G_f be an access graph for a fixed-track configuration D_f. If D_u is route-equivalent to D_f, then G_f is a topological subgraph of G_u.

Proof: The fixed-track access graph algorithm, Algorithm 4.1, can be mapped directly onto the unconstrained access graph algorithm, Algorithm 3.1, as discussed in Section 3.2. Algorithm

4.1 generates no arcs that would not also be created by Algorithm 3.1 if both were to process the same configuration. Since the number of nodes created in step 1 of the algorithm is the same, then a graph G_f generated by Algorithm 4.1 is a subgraph of a graph G_u generated by Algorithm 3.1 where both process route-equivalent configurations. Thus, G_f is a topological subgraph of G_u. □

We now give two related bounds on the size of the fixed-track access graph.

Theorem 4.1: Let n be the number of points in the routing configuration D, and c the number of segments routed in D. Then the number of arcs k in the fixed-track access graph G for D is

$$k \leq 3n-c-3 .$$

Proof: By Theorem 3.1, the number of arcs k' in the unconstrained access graph G' for D is exactly $3n-c-3$. Since the fixed-track graph G is a topological subgraph of G', then the number of arcs k in G is less than or equal to k'. Thus, $k \leq 3n-c-3$. □

$$k_{max} = \frac{3(2n-4)+4+2}{2} = 3n-3 .$$

4.3 Minimum-Length Paths in a Fixed-Track Configuration

A minimum-length feasible path may be found in a fixed-track configuration in the identical way as for the unconstrained case discussed in Chapter 3, Section 3.3.

As an example of determining a minimum-length path for a net in a fixed-track configuration, consider the configuration shown in Figure 4.8. Assume a net $N = (p_2, p_6)$ is to be routed.

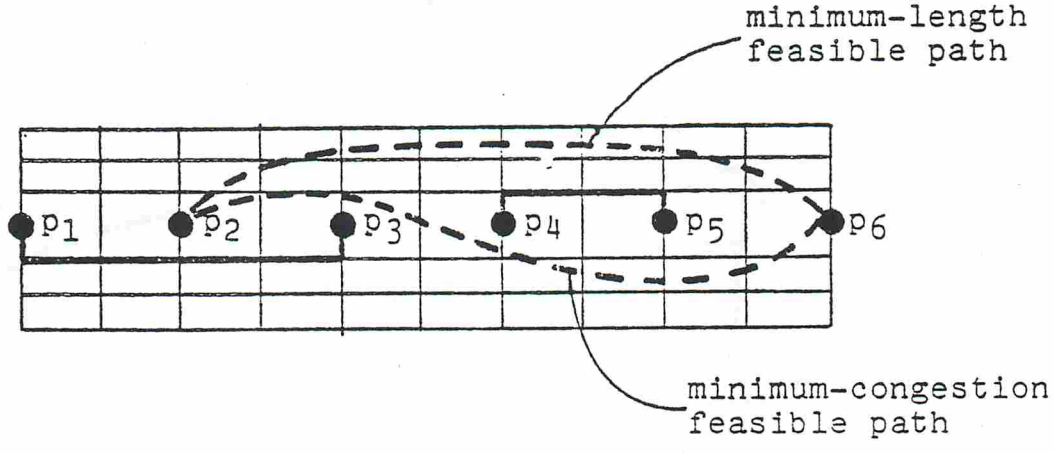


Figure 4.8. A fixed-track configuration illustrating the determination of a minimum-length and a minimum-congestion feasible path.

The fixed-track access graph for this configuration is shown in Figure 4.9. The heavy arcs represent a (in this case, the only) minimum-length path between nodes 2^+ and 6^+ as found by Algorithm 3.2. The path in the configuration represented by the path shown in Figure 4.9 is the minimum-length feasible path shown in Figure 4.8.

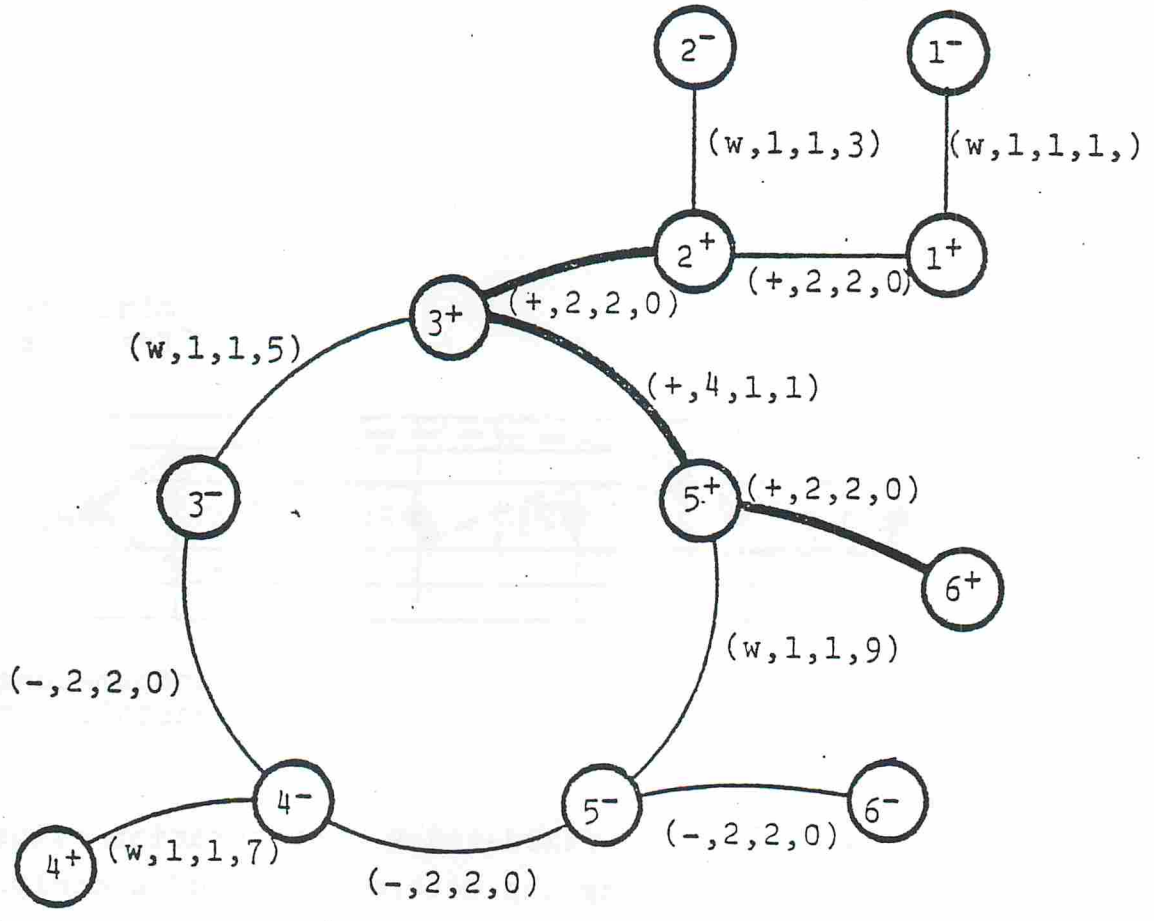


Figure 4.9. The access graph for the configuration shown in Fig.4.8. The heavy arcs represent the minimum-length path between nodes 6^+ and 2^+.

The following corollary determines the bounds on k in terms of the number of original points in D. This bound is not as tight as that given by Theorem 4.1.

Corollary 4.1: The bounds on the number of arcs k in the fixed-track access graph G for a routed configuration D with m original unidirectional points and |u| switching tracks between each pair of unidirectional points is

$$0 \leq k \leq 3m-3 .$$

Proof: Since there could possibly be no free tracks in D, the minimum number of arcs in G is 0. The maximum number of arcs is derived from the upper bound stated in Theorem 4.1. Let n be the number of points in D, c the number of segments routed in D, r the number of pseudo-points in D, and m the number of original points in D. By definition, $n = m+r$ and $r = 0$ if $c = 0$. Since $c \geq 0$,

$$\begin{aligned} k &\leq 3n-c-3 \\ &= 3r+3m-c-3 \\ &\leq 3(m-1) . \end{aligned}$$

□

Thus, the maximum size of the fixed-track access graph is a linear function of the number of unidirectional points in D.

One can also derive the result as stated for the upper bound in Corollary 4.1 if the unconstrained access graph containing the greatest node *degree* is found. The configuration generating this type of graph is the point line without any routed segments. The access graph for this configuration is shown in Figure 4.7.

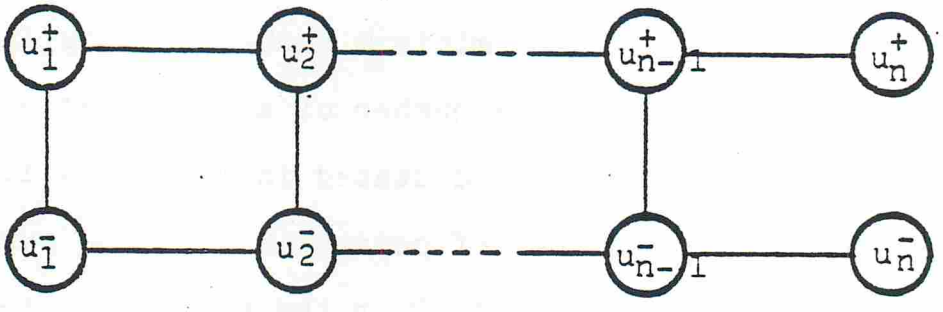


Figure 4.7. The access graph containing the most arcs for a configuration with n points. Arc labels are not shown for clarity.

Notice that two nodes have degree two, two nodes have degree one, and all other (i.e., 2n-4) nodes have degree three. Thus, there is a total degree of 3(2n-4)+2(2)+1(2). Since the number of arcs in a graph is one half the total degree, then

4.4 Minimum-Congestion Paths in a Fixed-Track Configuration

Routing *congestion* is defined to be the number of routed wires in a given board area. It is hypothesized that minimizing routing congestion as each new wire is added potentially increases the total number of wires that may be routed in a unidirectional configuration.

For the fixed-track case considered here, instead of minimizing congestion, the equivalent objective of maximizing the number of free tracks available for each feasible path is used. To be exact, we wish to obtain a path for a net N such that some function of the number of free horizontal tracks available for routing the sections of the path for N is maximized for any path in D for N. It is because of this goal that the parameter ρ for each arc in the fixed-track access graph was defined.

To motivate this section, consider once again the configuration shown in Figure 4.8. If the minimum-length feasible path is considered, the minimum number of tracks available for the feasible section in street S^+ is one. However, the minimum number of tracks available for the two feasible sections (one in street S^+ and one in S^-) of the minimum-congestion feasible path is two. This illustration also serves to show that minimum-congestion paths are not necessarily minimum-length paths (although they can be).

We will now show how to find a fixed-track minimum-congestion path.

Let $N = (s,t)$ be a two-point net, and consider the access graph G as created by Algorithm 4.1 for a configuration. Let the label of each section be (x,δ,ρ,t) where x is "+" or "-," δ is the length of the represented c-section, and ρ is the track capacity of the channel represented by the arc. The objective is to find a path in the access graph from nodes s^x to t^x consisting of arcs $a_1,a_2,\ldots$ with channel capacities $\rho_1,\rho_2,\ldots$ such that for any other path between s^x and t^x consisting of arcs $a_1,a_2,\ldots$ with channel capacities $\rho'_1,\rho'_2,\ldots$

$$\max_i(\rho_i) \leq \max_j(\rho'_j) .$$

Fortunately, Dijkstra's least-cost algorithm as presented in Chapter 2 can be easily modified to become a "min-max" algorithm sufficient for our purposes. In step 2 of the algorithm, replace the operation "replace $\ell(v)$ by $\min\{\ell(v),\ell(u_i)+w(u_i,v)\}$" with "replace $\ell(v)$ by $\min\{\ell(v), \max[\ell(u_i),w(u_i,v)]\}$." The rest of the algorithm remains as originally defined. The run-time complexity of the algorithm is unchanged from that given in Chapter 2.

The minimum-congestion path for a net $N = (s,t)$ in a configuration is found in a similar fashion as for a minimum-length path (Algorithm 3.2, Chapter 3) except that

the "min-max" modified Dijkstra's algorithm is used in step 2 with $w(u_i,v) = T^x - \rho(u_i,v)$ where ρ is the number of free tracks in the channel represented by the arc (u_i,v), and T^x is the number of tracks (free and occupied) in the street containing the channel.

As an example of finding a minimum-congestion path, consider the configuration shown in Figure 4.10. The access graph for this configuration is shown in Figure 4.11.

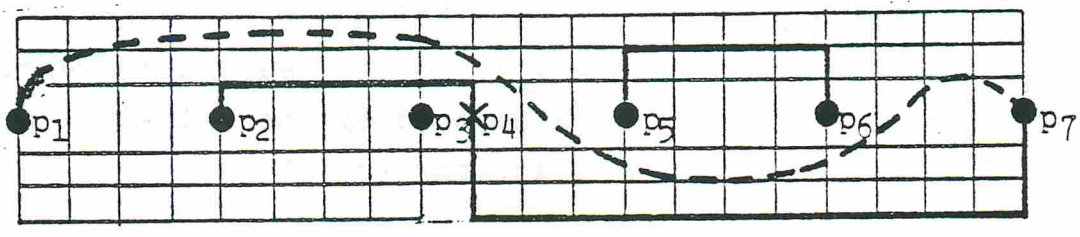


Figure 4.10. A configuration illustrating the determination of a minimum-congestion path. The dotted line represents a feasible minimum-congestion path for $N = (1,7)$.

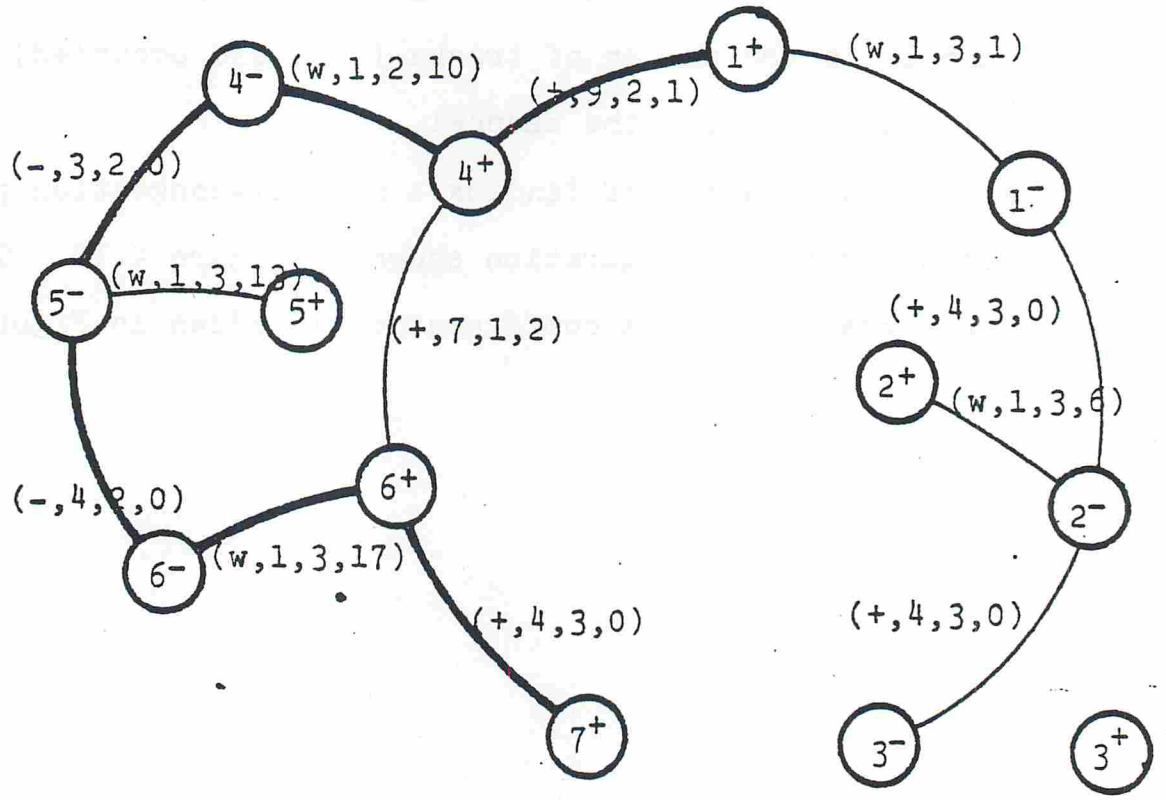


Figure 4.11. Access graph for the configuration in Fig.4.10. The heavy arcs represent the path found by the "min-max" modified algorithm (see text).

The "min-max" modified algorithm is executed on the access graph resulting in the path shown in Figure 4.11. Note that if the standard least-cost algorithm were executed instead, the path $(1^+,4^+,6^+,7^+)$ would have been found.

4.5 Fixed-Track Assignment

The last step in routing a wire in a unidirectional configuration is to assign a portion of a free track to each feasible section making up the feasible path. First, we define several related configurations.

Let D_i be a fixed-track configuration prior to the consideration of any feasible paths. If a feasible path for some two-point net N_j has been defined on D_i, then we call the configuration a *feasible routed configuration* D_i. After the path for N_j has been assigned to tracks, the new routed configuration is labeled D_i^*. We have that $D_{i+1} = D_i^*$. Thus, the routing problem is one of successively determining three configurations for each net N_j to be added (i.e., D, D′, and D^*). In the discussion that follows we consider a single two-point net N. Thus, the subscripts on D, D′, and D^* are not needed.

The procedures presented here are based upon the following assumptions:

1. all future nets that could be routed in D are equally likely to occur.
2. no routed paths are rerouted or reassigned to free tracks when routing net N.

The objective is to allocate tracks to feasible sections such that the probability of routing future nets is maximized. To do this required knowledge of future

channel capacities. To gain this information we must first construct (Algorithm 4.2) a partially arc-labeled graph G′ which reflects the existence of the feasible path but does not require information with regard to the tracks assigned to the feasible path. Given G′, and based upon an estimate of future track utilization, Algorithm 4.3 is used to assign the feasible path to tracks.

4.5.1 Generating a feasible access graph G′ from access graph G

Some of the arcs in G′ have incompletely specified arc weights because a feasible c-section in the channel represented by such an arc has not yet been assigned to a track. The number of vertices m′ in G′ is greater than the number of vertices m in G by an amount 2g where g is the number of switches in the feasible path in D′. Thus,

$$m' = m + 2g .$$

As will be shown later, the number of arcs k′ in G′ is

$$k+g-2 \leq k' \leq k+g .$$

The algorithm for creating the feasible access graph G′ from the access graph G involves splitting some of the nodes in G into two nodes and deleting and adding arcs to these nodes as necessary. As an illustration of the node splitting process, consider Figure 4.12.

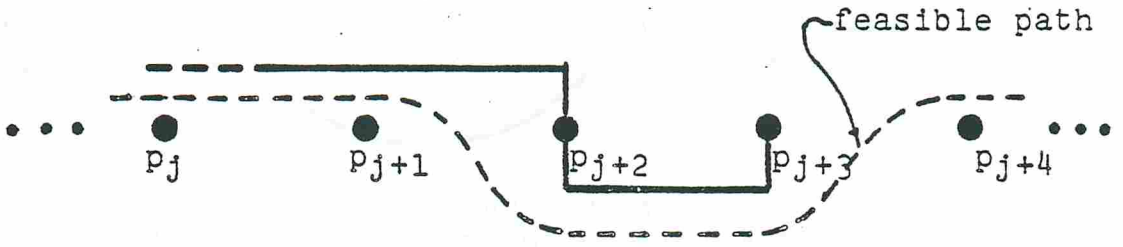
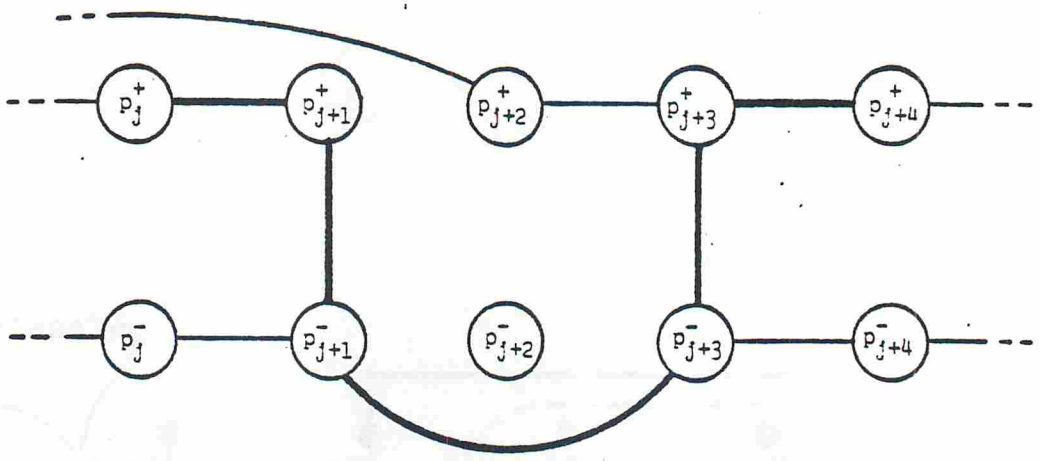


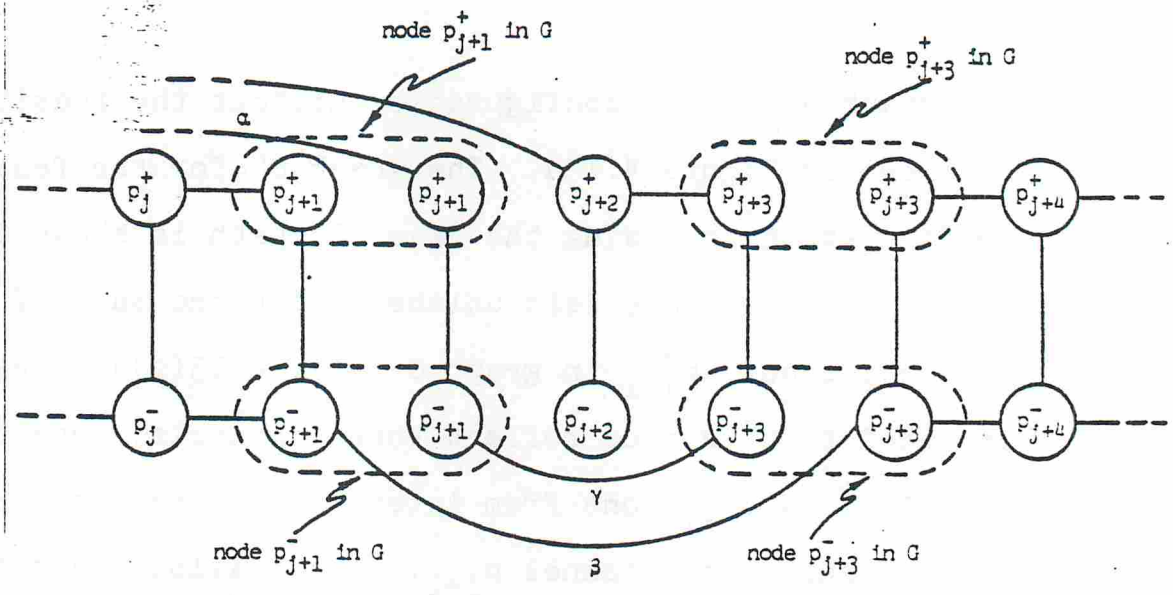
Figure 4.12. Part of a feasible configuration illustrating the creation of G′ from G.

The graph G for the configuration without the feasible path is shown in Figure 4.13a. The graph G′ for the feasible configuration including the feasible path is shown in Figure 4.13b. Arcs are left unlabeled for the sake of clarity.

Notice node p^-_{j+1} in graph G (Fig. 4.13(a)). The arcs attached to this node reflect three channels: one from interval p^-_j to p^-_{j+1}, one from interval p^-_{j+1} to interval p^-_{j+3}, and the switching channel p_{j+1}. The feasible path in Figure 4.12 splits the interval u_{j+1} into two intervals, u_{j+1} and $\hat{u}_{j+1}$ (the "hat" is an arbitrarily-chosen symbol used to differentiate the two new adjacent intervals). In fact, the feasible switching path in interval u_{j+1} defines a



(a) Graph G for the partial configuration shown in Fig.4.12. The heavy arcs reflect the path in Fig.4.12.



(b) Graph G′ derived from graph G where nodes p_{j+1}^+, p_{j+2}^-, p_{j+3}^+, and p_{j+3}^- in G are each split into two nodes. Arc a in G becomes arcs γ and β in G′, and arc α is added to G′.

Figure 4.13. Graphs G and G′ for the configuration shown in Fig. 4.12.

"feasible" pseudo-point, $\hat{p}_{j+1}$ (Fig.4.13(b)). The interval $\hat{u}_{j+1}$ is shown as the two nodes $\hat{p}^{+}_{j+1}$ and $\hat{p}^{-}_{j+1}$ in G'. We say these two nodes are *partners* of each other.

Now notice arc a in G (Fig.4.13(a)). This arc is reflected as arc γ in G' (Fig.4.13(b)). The adjacent node p^{-}_{j+1} in G is no longer valid when the feasible path is considered. Instead, node $\hat{p}^{-}_{j+1}$ becomes the adjacent node in G'. Also, a new arc β is added to reflect the outer channel of the feasible segment $(p^{-}_{j+1}, \hat{p}^{-}_{j+3})$.

In general, the arcs adjacent to all split nodes in G' are only partially labeled since the feasible path has not yet been assigned to tracks. Algorithm 4.4 presented later in this chapter completes the labeling of these arcs as a result of assigning the feasible path to tracks.

4.5.1.1 Algorithm 4.2. Create a feasible access graph G'

Purpose: This procedure creates a feasible fixed-track access graph G' from a fixed-track access graph G and a description of a feasible path P defined on G.

Method: Each node in the path P in G is considered in turn. If the node being considered is adjacent to a switching arc in P, then it is replicated and marked with a hat (^). Arcs are deleted and added as the scan continues along the path until the end of the path is reached. The resulting graph is G'. In this procedure we use the

concept of *node partners* where $v^{\bar{x}}$ is the partner of v^x. For example, if $v^x = v^-$, then $v^{\bar{x}} = v^+$, and if $v^x = v^+$, then $v^{\bar{x}} = v^-$. This algorithm uses two pointers: z^x which is always pointing to the current node being processed, and g which points to the first node of each arc sequence representing a feasible section in the configuration. Once such an arc sequence is found, g points to the first node in the sequence and z^x points to the last node. For example, consider the path P_1 shown in Figure 4.14. Switching arcs are labeled "w". As the procedure executes, g will take on the values 2^+, 0, 4^-, 0, and 7^+ successively.

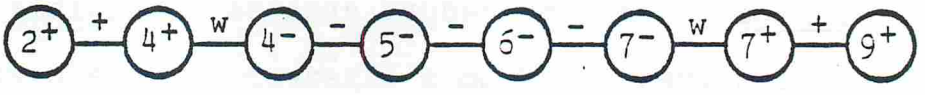


Figure 4.14. Path P_1 illustrating the values taken on by parameter g as Algorithm 4.2 is executed.

Input: An access graph G and a path P defined on G.

Output: Feasible access graph G'.

Procedure:

Step 1. (initialization) Set G' = G. Let the first node in path P be s^x. Since interval s^x-1 in the feasible configuration represented by G' is no longer accessible to interval s^x, delete arc $(s^x\text{-}1, s^x)$ from G'. Let $g = s^x$. If s^x is adjacent to a switching arc in P, then set $z^x = s^x$ and go to step 3.

Step 2. (begin main loop) Let z^x be the next node in P. If z^x is the final node in P, then go to step 6.

Step 3. (node and switching arc creation)

(a) If z^x is adjacent to a switching arc in P, then create a node $\hat{z}^x$ in G' and go to part (b) of this step. Otherwise, let u^x be the predecessor node of z^x in P. Replace the third parameter in the label of arc (u^x, z^x) with the symbol "Σ". Go to step 2.

(b) If the node partner $\hat{z}^{\bar{x}}$ does not exist, go to part (c) of this step. Otherwise, add a new switching arc $(\hat{z}^{\bar{x}}, \hat{z}^x)$ to G'. Label this arc (w,1,"Σ", "α"). Relabel arc

$(z^{\bar{x}}, z^x)$ by $(w, 1, "\Sigma", \alpha_z)$. (The symbols in quotes are literally the arc labels. They are replaced by actual numerical values in a later procedure.) α_z is the switching track number of point p_z in the configuration represented by G.

(c) If $g = 0$ set $g = z^x$ and go to step 5.

Step 4. (create the section arc produced by feasible segment)

(a) If $g = s^x$, $s \neq 1$, and $g < z$ then add arc $(g-1, \hat{z}^x)$ and label it $(x, "\Delta", "\Sigma", "T")$.

(b) If $g = s^x$, $s \neq n$, and $g > z$ then add arc (z^x, g) and label it $(x, "\Delta", "\Sigma", "T")$.

(c) If $g \neq s^x$ and $g < z$ then add arc $(g, \hat{z}^x)$ and label it $(s, "\Delta", "\Sigma", "T")$.

(d) If $g \neq s^x$ and $g > z$ then add arc $(z^x, \hat{g})$ and label it $(x, "\Delta", "\Sigma", "T")$.

(e) Set $g = 0$.

Step 5. (end of main loop)

(a) Let u^x be a node adjacent to z^x in G' where $u < z$. Replace the second and third arc label parameter of arc (u^x, z^x) with "Δ" and "Σ" respectively.

(b) Let v^X be a node adjacent to node z^X in G' where $v > z$. Since interval z is no longer accessible to interval v in street S^X in the feasible configuration represented by G', delete arc (z^X, v^X). Let the value of the fourth parameter in the label of the deleted arc be T'.

(c) If v^X is the last node in P_1 go to step 2. Otherwise, the new interval $\hat{z}$ is accessible to v in street S^X so add an arc $(\hat{z}^X, v^X)$ to G' and label it (x,"Δ","Σ",T'). Go to step 2.

Step 6. (termination and exit)

(a) If z^X is adjacent to a switching arc in P then go to step 7. Otherwise, delete arc (z^X-1, z^X) in G'.

(b) If $g = s^X$, $s \neq 1$, $z \neq n$, and $g < z$ then add arc $(g-1, z^X)$ and label it (x,"Δ","Σ", "T").

(c) If $g = s^X$, $s \neq n$, $z \neq 1$, and $g > z$ then add arc (z^X-1, g) and label it (x,"Δ","Σ", "T").

(d) If $g \neq s^x$, $z \neq n$, and $g < z$ then add arc (g, z^x) and label it (x,"Δ","Σ","T").

(e) If $g \neq s^x$, $z \neq 1$, and $g > z$ then add arc $(z^x\text{-}1, \hat{g})$ and label it (x,"Δ","Σ","T").

(f) Exit algorithm.

Step 7. (This step is entered only if the last node in P is adjacent to a switching arc.)

(a) Create a new node $\hat{z}^x$ in G'. Add an arc $(\hat{z}^{\bar{x}}, \hat{z}^x)$ to G' and label it (w,1,"Σ", α_z). α_z is the switching track number of point p_z in the configuration represented by G.

(b) Delete arc $(z^x\text{-}1, z^x)$ and add arc $(z^x\text{-}1, \hat{z}^x)$. Label it (x,"Δ","Σ","T").

(c) Let v^x be a node in G' adjacent to node z^x where $v > z$ (there may be none). Delete arc (z^x, v^x). Let T' be the value of the fourth parameter of the label of arc (z^x, v^x). Add arc $(\hat{z}^x, v^x)$ to G' and label it (x,"Δ","Σ",T'). Exit algorithm. □

4.5.1.2 An example of the operation of Algorithm 4.2

The configuration shown in Figure 4.15 and its corresponding access graph as shown in Figure 4.16 will be used for this example. The path P is shown in heavy arcs in the

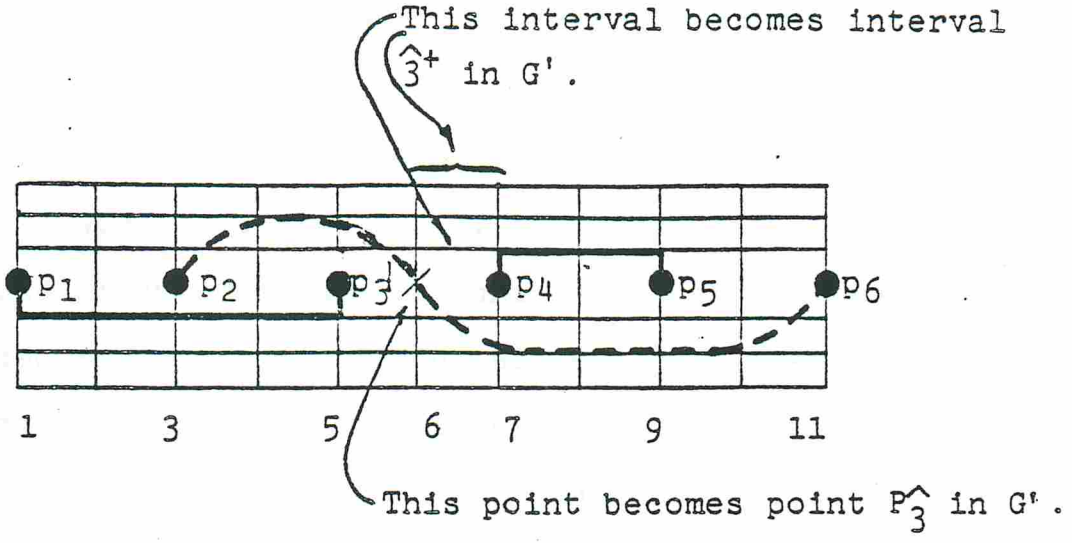


Figure 4.15. A feasible configuration illustrating Algorithm 4.2.

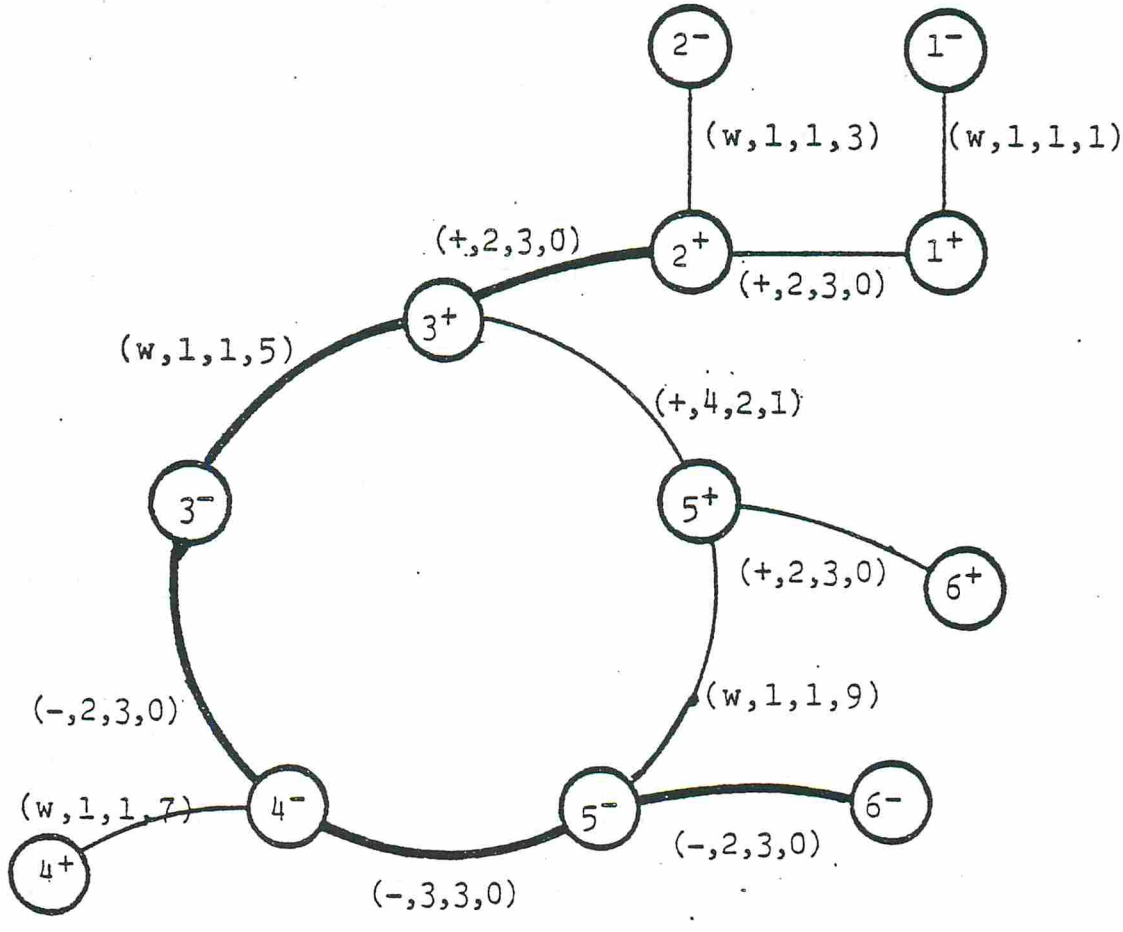


Figure 4.16. The access graph for the configuration in Fig.4.15.

graph. It corresponds to the minimum-congestion path shown in Figure 4.15.

Figure 4.17 illustrates the major successive steps taken by Algorithm 4.2. The path P is shown separately from the graph for clarity. The numbers in the squares represent the order in which operations are performed by the algorithm. Arc labels are shown only as they provide useful information.

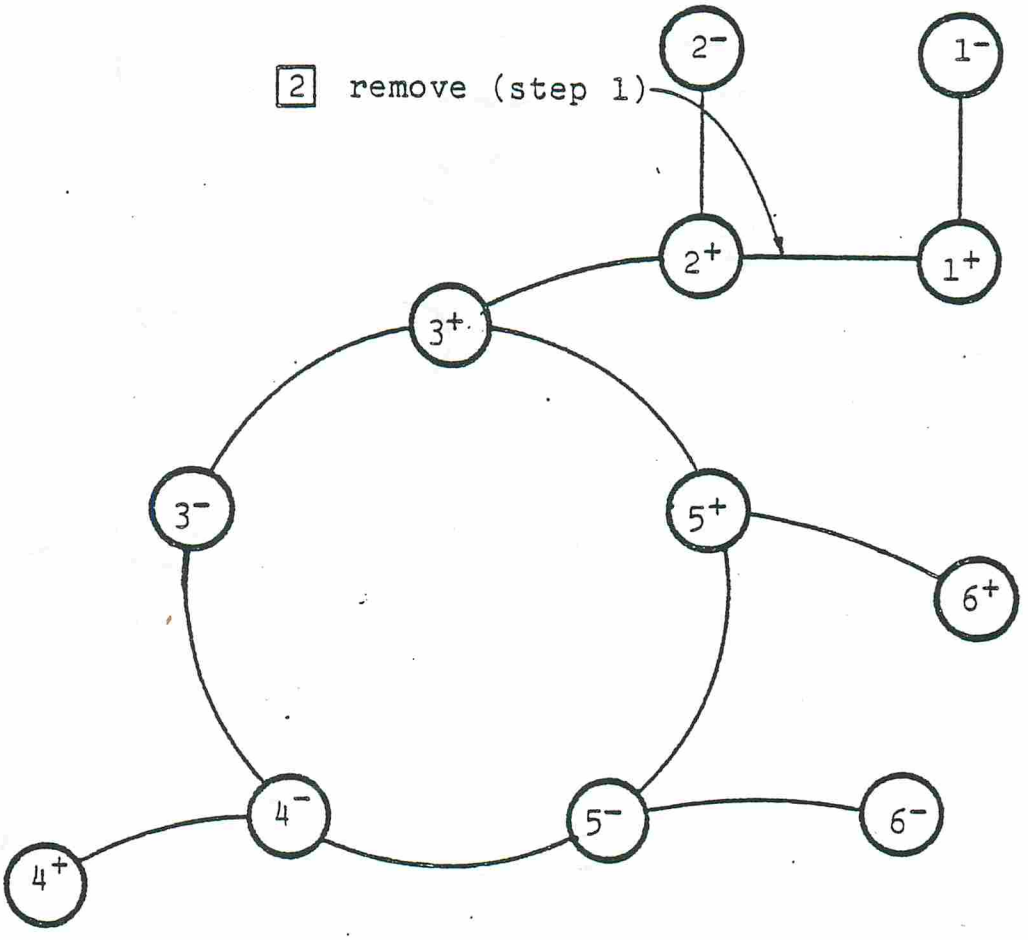
4.5.1.3 Analysis of Algorithm 4.2 and graph G'

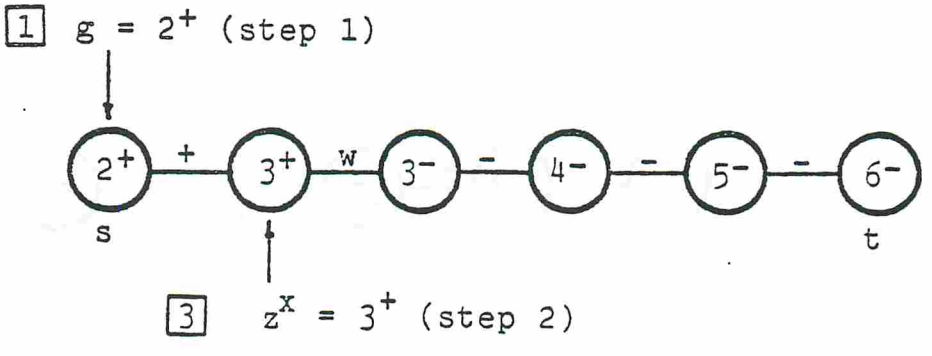
Let graph G have m nodes and k arcs, and let G' have m' nodes and k' arcs. Let a path P be defined on G. It is obvious that every switching arc in P implies two switching arcs in G'. Thus, each switching arc in P contributes two more nodes and one more arc in G' than in G. Algorithm 4.2 does not destroy any nodes. Therefore, if P has g switching arcs, then

$$m' = m+2g .$$

The number of arcs k' in G' is given by the following theorem.

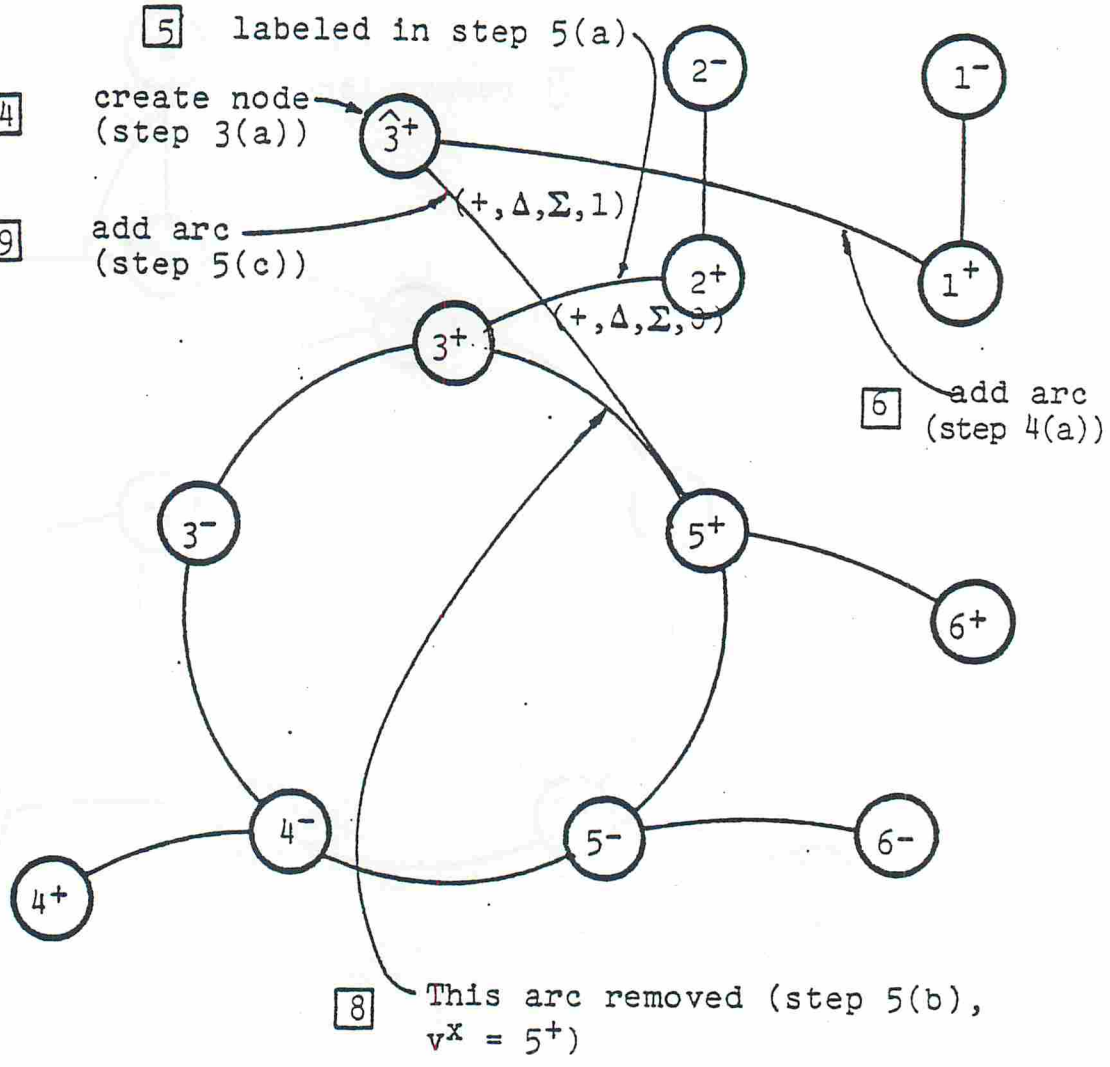
Theorem 4.2: Let an access graph G for a routed configuration D have k arcs and m nodes. Let a net N define a graph G' obtained from G by Algorithm 4.2 and let G' (the access graph for the feasible routed configuration D') have k' arcs and

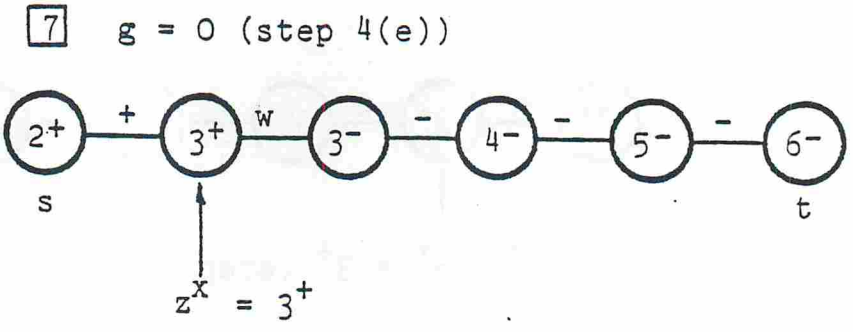




(a) The initial steps performed by Algorithm 4.2.

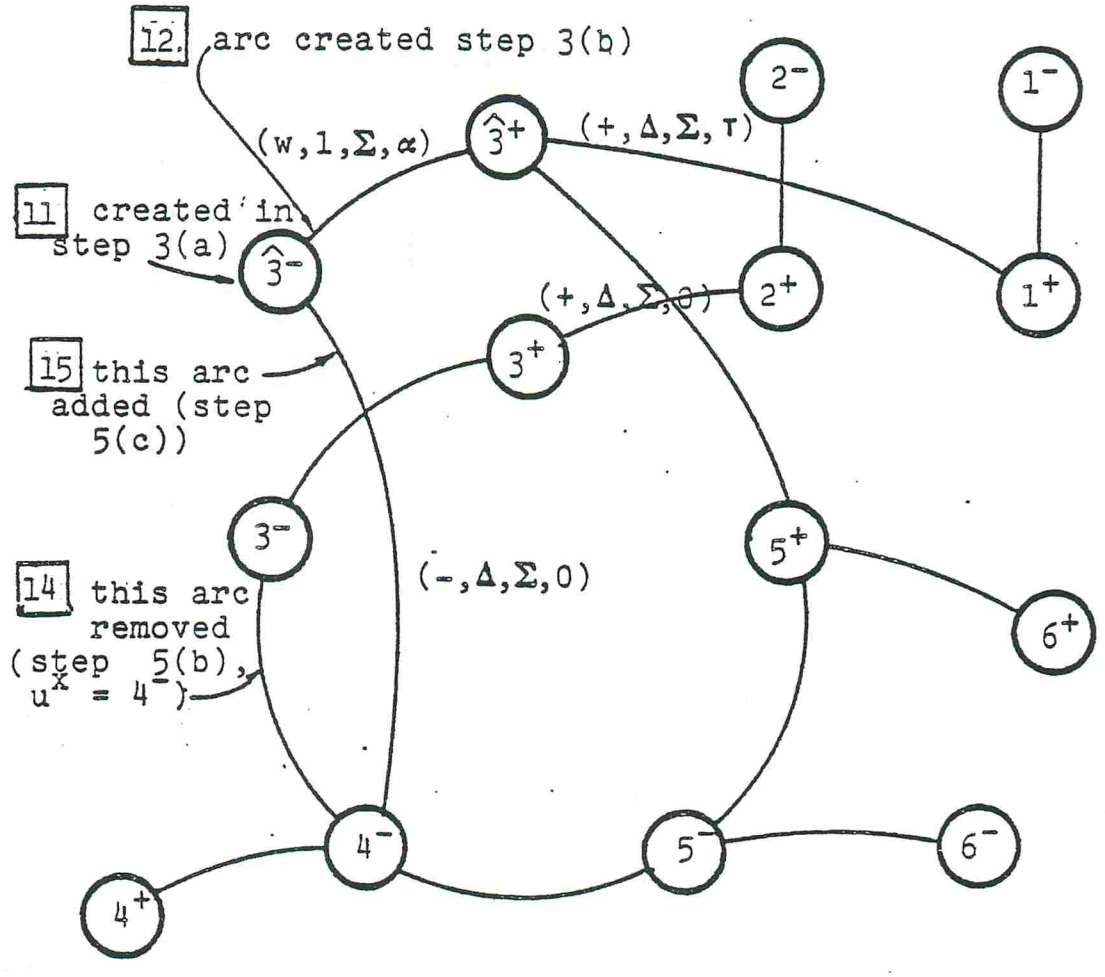
Figure 4.17. An example of the steps taken by Algorithm 4.2 to obtain graph G′ given graph G as shown in Fig. 4.16.

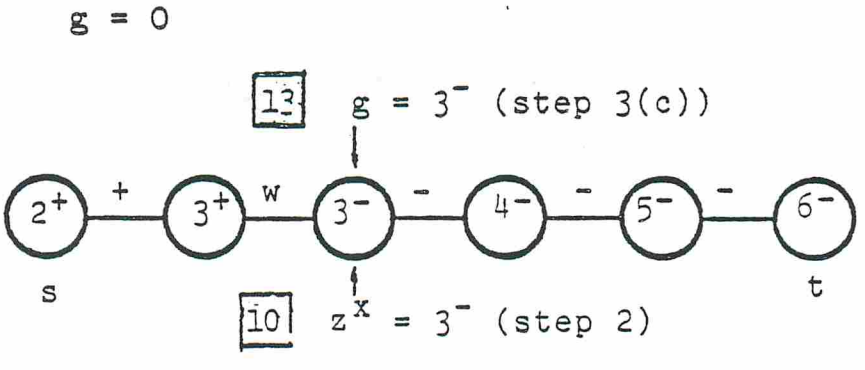




(b) First pass through the main loop (i.e., steps 2-7) of Algorithm 4.2.

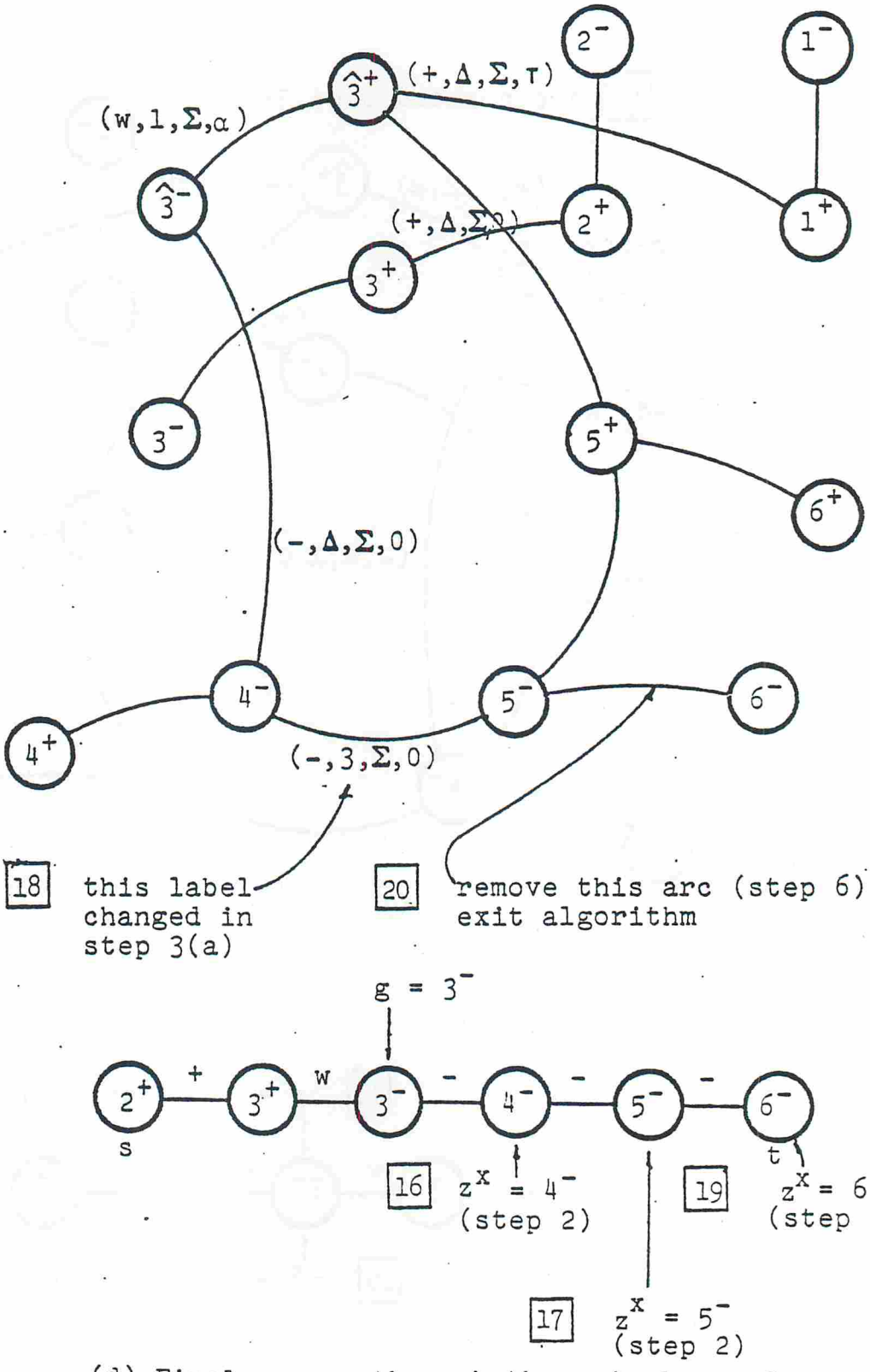
Figure 4.17 (continued)





(c). Second pass through the main loop of Algorithm 4.2.

Figure 4.17. (continued)



(d) Final passes through the main loop of Algorithm 4.2. Major activity is moving z^x pointer from node-to-node until last node is found. Final graph is G'.

Figure 4.17. (continued)

m' nodes. Also, let the path P representing the route of net N in G contain a total of h arcs of which g are switching arcs. Then,

$$k+g-2 \leqq k' \leqq k+g .$$

Proof: Let D have a set C of n unidirectional points. Each segment of the feasible path in D′ for N has the following possible combinations of end points in D:

1. both end points are in C.
2. only one end point is in C.
3. neither end point is in C.

Consider each of these combinations in turn as they impact the generation of arcs in G′. Let primed nodes be the additional nodes in G′ not in G.

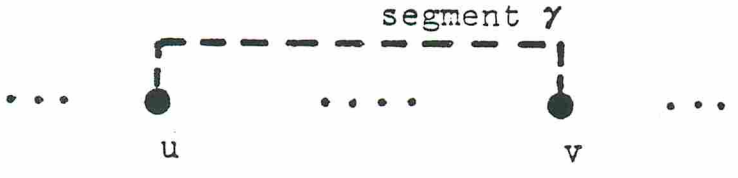


Figure 4.18. Both segment end points in C.

1. Both end points in C

Consider Figure 4.18. Note that segment γ implies the following in the street occupied by γ:

a. an arc (u-1,u) exists in G unless u = 1.

b. arc (u-1,u) does not exist in G'.

c. an arc (v-1,v) exists in G.

d. arc (v-1,v) does not exist in G'.

e. an arc (u-1,v) will be in G' unless u = 1 or v = n or both.

f. arc (u-1,v) is not in G.

Items a-f represent all ways G and G' may differ for a feasible segment with both end points in C. Thus, we see that if u = 1, or if both $u \neq 1$ and $v \neq n$, then G' contains one less arc than G. If $u \neq 1$ but v = n, then G' contains two less arcs than G.

2. Left-most end point in C

Figure 4.19 illustrates this configuration. The following differences exist between G and G' in the street in which γ lies:

a. an arc (u-1,u) exists in G unless u = 1.

b. an arc (u-1,u) does not exist in G'.

c. if an arc (v,w), w > v, exists in G, it exists as (v',w) in G'.

d. an arc (u-1,v′) exists in G′ unless u = 1.

e. arc (u-1,v) does not exist in G.

Differences a-e indicate that G′ contains the same number of arcs as G for this case.

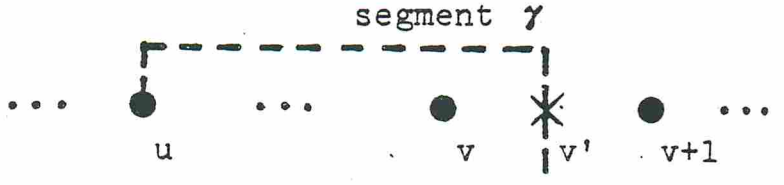


Figure 4.19. The left-most end point of segment γ is in C. (v′ is not in C.)

3. Right-most end point in C

By symmetry with case 2, the same conclusion is reached.

4. Neither end point is in C

Figure 4.20 represents this situation. The following differences exist between G and G′ in the street in which γ lies:

a. arc (u,v') exists in G'.

b. if arc (u,w) $w > u$, exists in G, then arc (u',w) exists in G'.

c. if arc (v,w) $w > v$, exists in G, then arc (v',w) exists in G'.

Figure 4.20. Neither end point of segment γ is in C. (Neither u' nor v' is in C.)

For this configuration, G' contains the same number of arcs as G. Thus, by considering the applicability of each configuration to the entire path P, and including the additional switching arc in G' for each switching arc in P, we get

$$k+g-2 \leq k' \leq k+g \, . \qquad \square$$

The proof of Theorem 4.2 is constructive, and is the basis for Algorithm 4.2.

From observation of Algorithm 4.2, it is obvious that the timing complexity of the algorithm is no greater than linear order. This is shown in the following theorem.

Theorem 4.3: Algorithm 4.2 is of complexity $\hat{O}(n)$ where n is the number of points in D.

Proof: Algorithm 4.2 executes a constant number of steps for each arc examined in path P, and each arc is examined in turn from the first to the last without retrace. Thus, the execution time of the algorithm is directly porportional to the length of P. P can be no longer than the number of arcs in G which is directly proportional to the number of points in D as shown by Theorem 4.1. Thus, the execution time of Algorithm 4.2 is directly proportional to the number of points in D. □

4.5.2 Assigning a track to a feasible segment

Consider the two segments shown in Figure 4.21. For segment 1, three tracks are available for any future paths. However, segment 2 completely blocks the street. Thus, we see that the track selected for a current feasible segment can impact the routability of future nets.

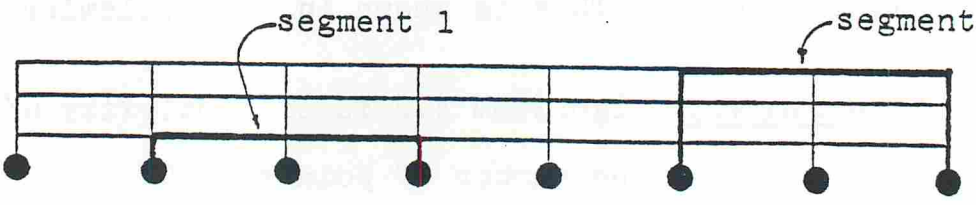


Figure 4.21. Track assignments have an impact on future paths.

Assume a feasible access graph G′ containing a feasible path for a two-point net N exists for a given feasible routed configuration D′. Each feasible segment in the path is assigned to an unused track using the algorithm to be presented here. By construction of G′, such a track always exists. The track assignment algorithm assigns a free track to a feasible section in the channel occupied by the section such that the probability of having free tracks available for future paths is maximized. This is accomplished by determining the probability P_1 that a future segment is likely to occupy the outer channel of a feasible segment, and the probability P_2 that a future segment is likely to occupy the inner channel of the same feasible segment (see Figure 4.22). If we assume there are

$T = t_1 + t_2 + 1$ free tracks available for the feasible segment, then the free track τ to be assigned within the channel to the segment is determined as follows.

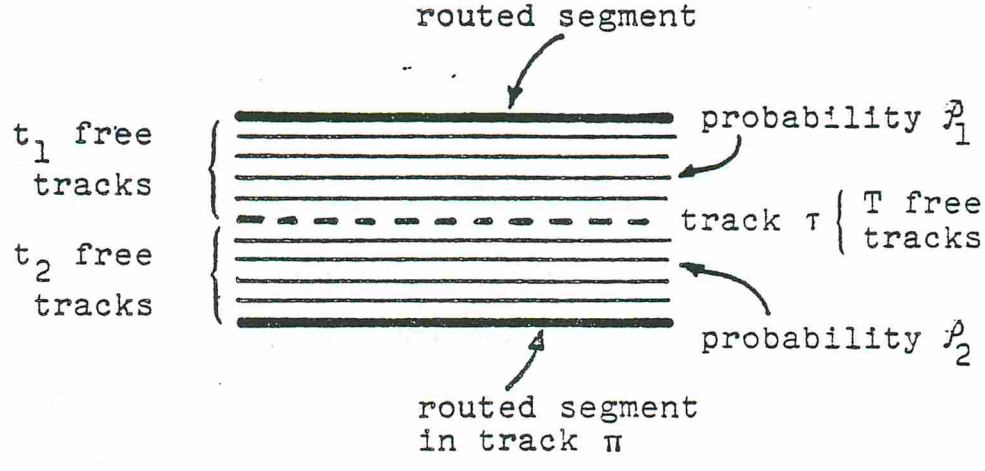


Figure 4.22. A channel in a configuration illustrating the assignment of a free track to a feasible section.

Let $\frac{t_1}{t_2} = \frac{\rho_1}{\rho_2}$. Thus

$$t_1 = \frac{t_2 \rho_1}{\rho_2} \qquad (1)$$

Since $T = t_1 + t_2 + 1$ then, from (1), $T = \frac{t_2(\mathcal{P}_1 + \mathcal{P}_2)}{\mathcal{P}_2} + 1$ or,

$$t_2 = \frac{(T-1)\mathcal{P}_2}{\mathcal{P}_1 + \mathcal{P}_2} \tag{2}$$

Let $t = t_2 + 1$. From (2), $t = \frac{(T-1)\mathcal{P}_2}{\mathcal{P}_1 + \mathcal{P}_2}$.

$$= \frac{\mathcal{P}_2 T + \mathcal{P}_1}{\mathcal{P}_1 + \mathcal{P}_2} \tag{3}$$

Since t must be an integral value, $t = \left\{ \frac{\mathcal{P}_2 T + \mathcal{P}_1}{\mathcal{P}_1 + \mathcal{P}_2} \right\}$ where $\{x\}$ denotes x rounded to an integer value.

In the track assignment procedure described here, instead of determining the probability of a future segment occurring in a channel, the related concept of determining the number Ω of equally-likely future 2-point nets that could traverse the channel is used. If Ω_1 nets could be routed in the outer channel of a feasible section γ, and Ω_2 nets could be routed in the inner channel of γ, then we let $\Omega_1 = \mathcal{P}_1$ and $\Omega_2 = \mathcal{P}_2$ in expression (3) obtaining (for integral t)

$$t = \left\{ \frac{\Omega_2 T + \Omega_1}{\Omega_1 + \Omega_2} \right\} .$$

Assume the inner routed segment bounding the channel is assigned to track π. Then, we assign track τ to the feasible section where

$$\tau = \pi + t$$

$$= \pi + \left\{ \frac{\Omega_2 T + \Omega_1}{\Omega_1 + \Omega_2} \right\}. \qquad (4)$$

As an example, consider the feasible configuration shown in Figure 4.23. The question is: do we assign section γ to track 3 or 4? Using a procedure to be given later (Algorithm 4.3) we calculate Ω_1 to be 4 and Ω_2 to be 1. The four potential future nets contributing to Ω_1 are (p_1,p_7), (p_1,p_8), (p_2,p_7), and (p_2,p_8). The only potential future net which contributes to Ω_2 is (p_2,p_6).

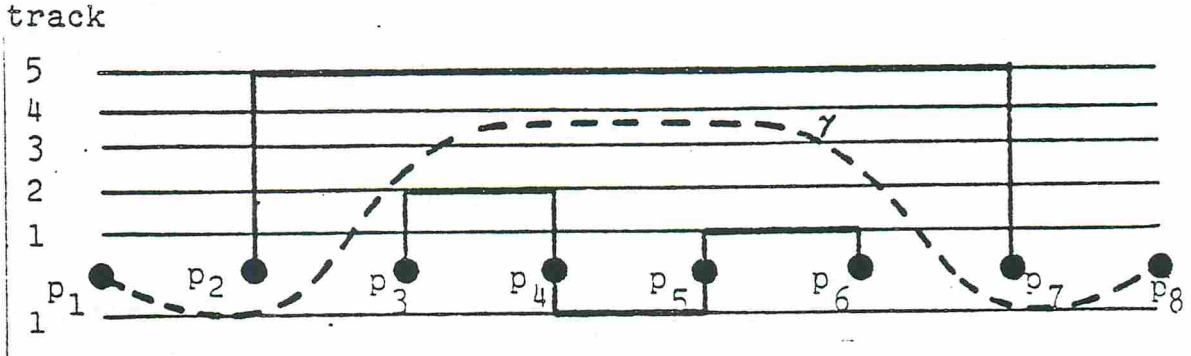


Figure 4.23. A feasible configuration illustrating the concept of track assignment in a channel.

4.5.2.1 Algorithm 4.3. Net counting procedure

Purpose: This procedure determines the number of two-point nets any one of which could be routed in a given channel ψ.

Method: Using the feasible access graph G′ as found by Algorithm 4.2, the number of two-point nets that can be routed is found such that at least one path for each net contains an arc in channel ψ. Only nodes representing uncovered points in the feasible routed configuration D′ are considered.

Input: A feasible access graph G′ for a feasible routed configuration D′, and a channel ψ represented as arc x in G′.

Output: The number Ω of nets that may be routed in channel ψ.

Procedure:

Step 1. Let $\bar{G}'$ be the connected subgraph in G′ containing arc x. Find the biconnected component[1] H in $\bar{G}'$ containing arc x. Let set V consist of the nodes of H.

[1]A biconnected component of a graph G is a maximal subgraph G′ of G such that G′ is a single isolated node, or every node in G′ has degree two or greater. For more information regarding biconnected components and how to find them, see Aho, Hopcroft, and Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley Publishing Co., 1974, pp. 179-186.

Step 2. Let H' be the graph $\bar{G}'$ with H removed. Separate H' into its components $H'_1, H'_2, \ldots, H'_k$, and let V'_i be the node set of H'_i.

Step 3. Remove from V and all V'_i, $i = 1,2,\ldots,k$, any node v representing a covered point in D'.

Step 4. Let n be the size of V and n'_i the size of V'_i. Then the number Ω of two-point nets, any one of which could be routed in channel ψ, is

$$\Omega = \frac{n(n-1)}{2} + (n-1)\sum_{i=1}^{k} n'_i + \sum_{i<j}^{k} n'_i n'_j . \qquad \square$$

4.5.2.2 Discussion and example of use of Algorithm 4.3

Consider the connected graph $\bar{G}' \subset G'$ where G' is the feasible access graph for some feasible configuration D' as shown in Figure 4.24. If a point p_i in D' is the end point of a segment (routed or feasible) in street S^x, then node i^x in $\bar{G}'$ is shown as a square. Otherwise, the node i^x is shown as a circle. In the discussion that follows, we are only interested in points in D' which could be end points of future nets. Thus, we will concentrate on circled nodes in $\bar{G}'$.

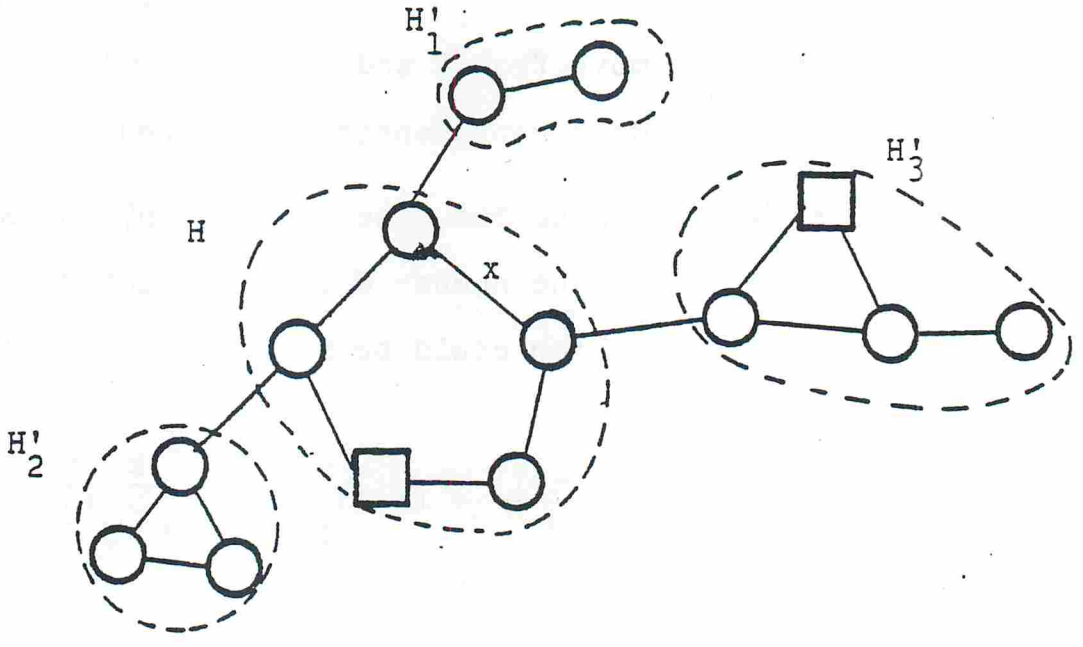


Figure 4.24. An example graph configuration illustrating the operation of Algorithm 4.3.

Divide $\bar{G}'$ into a biconnected component H and the three components H'_1, H'_2, and H'_3. Each component H'_i is connected to H by a single arc called a *bridge*. It is obvious that there are three kinds of paths in $\bar{G}'$ that contain arc x:

1. the paths with end points entirely in H;
2. the paths with one end point in H and the other in H'_i; and
3. the paths with one end point in H'_i and the other in H'_j, $i \neq j$.

These three types of paths give rise to the three terms in the expression for Ω_i in step 4 of the algorithm. Since we are interested in circle nodes only, and there are n circle nodes in the biconnected component H, then there are $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs of circle nodes in H, each of which is the end point of at least one path containing arc x.

Assume there are n'_i circle nodes in H'_i. Then there are $n \cdot n'_i - n'$ pairs of nodes (u,v) with u in H and v in H'_i that define paths which contain arc x since the single circle node adjacent to the bridge between H and H'_i does *not* define a path containing arc x if the other end of the path is in H'_i. Thus, considering all H'_i, the number of node pairs which define at least one path containing arc x is $(n-1) \sum_{i=1}^{k} n'_i$ where one of the nodes of each pair is in H.

Finally, if we assume that both end nodes of each path are in mutually exclusive components H'_i and H'_j then there exists $n'_i \cdot n'_j$ pairs of nodes where each pair defines at least one path in $\bar{G}'$ containing arc x. If we consider all k components, then $\sum_{i,j:i<j} n'_i \cdot n'_j$ paths are defined. Thus, the total number Ω of node pairs in G' where each node pair defines at least one path containing arc x is as shown in step 4 of the algorithm.

As an example of the use of Algorithm 4.3, consider Figure 4.25. Assume we are interested in determining the

number of possible paths in both the outer and inner channels of segment γ. The feasible access graph G' is as shown in Figure 4.26. In this case, G' is connected, so $\bar{G}'$ in step 1 of Algorithm 4.3 is the same as G'. The labels of arcs are not shown for the sake of simplicity.

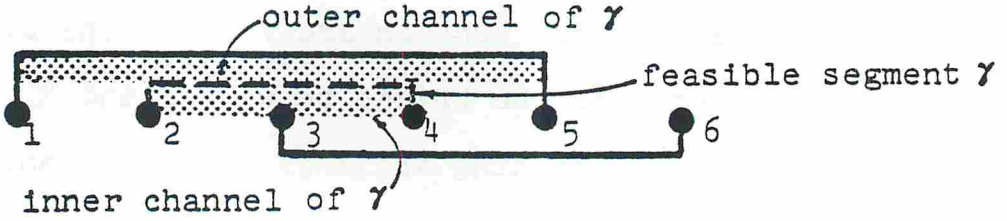


Figure 4.25. Example of feasible routed configuration illustrating use of Algorithm 4.3.

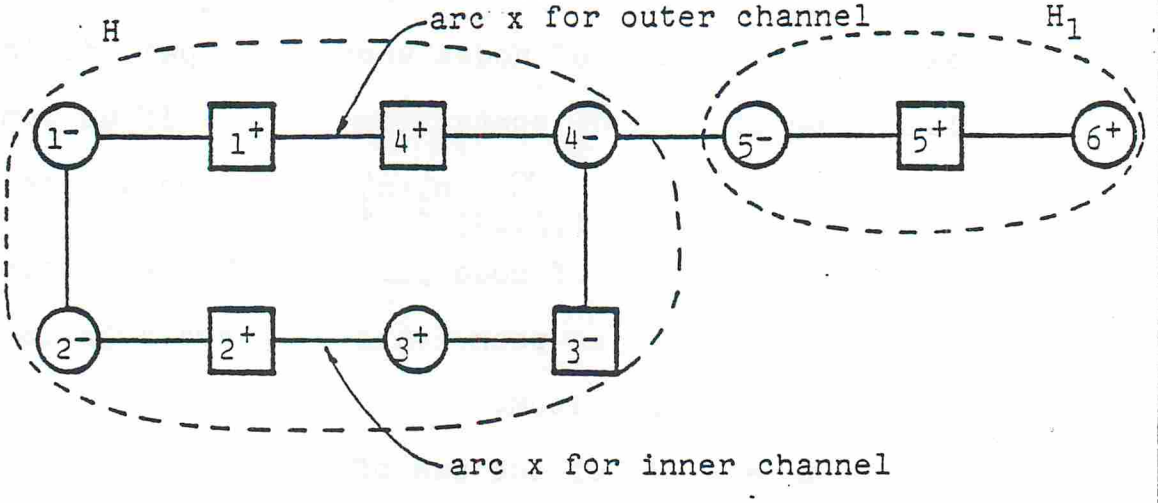


Figure 4.26. Feasible access graph for feasible routed configuration shown in Fig. 4.25.

The square nodes in Figure 4.26 represent points in Figure 4.25 that are covered. From Figure 4.26, we see there are 4 circle nodes in H and 2 circle nodes in H_1. Thus, $n = 4$ and $n_1' = 2$. By step 4 of Algorithm 4.3, the number of possible future nets N_1 in the outer channel is

$$\Omega_1 = \frac{4(3)}{2} + 3(2) + 0$$

$$= 12 .$$

For this example, since the arc x for the inner channel is in the same biconnected component as for the outer channel, the number of possible paths Ω_2 for the inner channel is the same as Ω_1, i.e., $\Omega_2 = 12$.

4.5.2.3 Timing of Algorithm 4.3

Since Algorithm 4.3 is a straightforward procedure without branch statements, we can determine the time necessary to do each step and then add the results to get an expression for the time complexity of the algorithm. To get $\bar{G}'$ from G' requires $\partial(e)$ steps (see Aho [28], p. 176) where $\bar{G}'$ has e edges. $\partial(e)$ operations are also required to determine the biconnected component H.

Thus, step 1 requires no more than $\partial(e)$ operations. Since the maximum number of nodes n in $\bar{G}'$ is a linear function of the number of edges e (see Corollary 4.1), step 1 is order $\partial(n)$.

Step 2 also requires $\mathcal{O}(n)$ operations to identify the components of H'. The actual process of creating the components is done as a byproduct of step 1. Step 3 requires a single examination of each node in $\bar{G}'$ to determine if the node is covered or not. Thus, step 3 is $\mathcal{O}(n)$. Finally, step 4 requires two multiplications and one subtraction for the first term in calculating N. The second term requires k additions, one subtraction, and one multiplication. The third and final term requires $\frac{(k^2-k)}{2}$ additions and the same number of multiplications. Thus, assuming all operations take unit time, step 4 requires k^2+4k+5 operations. The value of k will not be analytically considered here, but experience shows it is usually much smaller than n. However, in the worst case, k approaches $\frac{n}{2}$. Thus, step 4 requires up to $\frac{n^2}{4} + 2n + 5$ operations. Step 4 appears to be the predominant factor in determining the total time complexity of the algorithm. Thus, Algorithm 4.3 is of complexity $\mathcal{O}(n^2)$. However, experience shows that this algorithm normally behaves as an $\mathcal{O}(n)$ algorithm since k is usually very small.

4.5.2.4 Switching track selection

With Algorithm 4.3 available we can now assign each feasible section and switch represented by path P to appropriate tracks in D. We can also fix numeric values to all Δs and Σs in G' so that G' becomes the updated access

graph G^* for the routed configuration D^*.

In this section we describe how the switching arcs in path P are assigned to switching tracks in D'. Let the nodes in P be sequentially labeled $\beta_1^{x_1}, \beta_2^{x_2}, \ldots$, and let the arcs be sequentially labeled $\sigma_1, \sigma_2, \ldots$. Thus arc σ_1 is adjacent to nodes $\beta_1^{x_1}$ and $\beta_2^{x_2}$, σ_2 is adjacent to $\beta_2^{x_2}$ and $\beta_3^{x_3}$, etc. We scan path P for each switching arc in turn. Let σ_i be such a switching arc. Then there exists a switching arc $(\beta_i^{x_i}, \beta_i^{\bar{x}_i})$ with an undefined channel size Σ_i, and a switching arc $(\hat{\beta}_i^{x_i}, \hat{\beta}_i^{\bar{x}_i})$ with an undefined channel size $\hat{\Sigma}_i$. Let point β_i in D' be located at α_{β_i}. Pseudo-point $p_{\hat{\beta}_i}$ should then be located in the unoccupied switching track located at

$$\alpha_{\hat{\beta}_i} = \left\{ \frac{\Omega_2 |\alpha_{\beta_{i+1}} + \alpha_{\hat{\beta}_i} - 1| + \Omega_1}{\Omega_1 + \Omega_2} \right\} + \alpha_{\beta_i}$$

where Ω_1 and Ω_2 are obtained by using Algorithm 4.3; once for Ω_1, where x in the algorithm is arc $(\beta_i^{x_i}, \beta_i^{\bar{x}_i})$, and once for Ω_2 where x in the algorithm is arc $(\hat{\beta}_i^{x_i}, \hat{\beta}_i^{\bar{x}_i})$. Figure 4.27 illustrates the notation used thus far.

The value of Σ_i is thus $|\alpha_{\hat{\beta}_i} - \alpha_{\beta_i}| - 1$ and $\hat{\Sigma}_i = |\alpha_{\hat{\beta}_i} - \alpha_{\beta_{i+1}}| - 1$. If either Σ_i or $\hat{\Sigma}_i$ (or both) is zero, then delete the respective switching arc in G'.

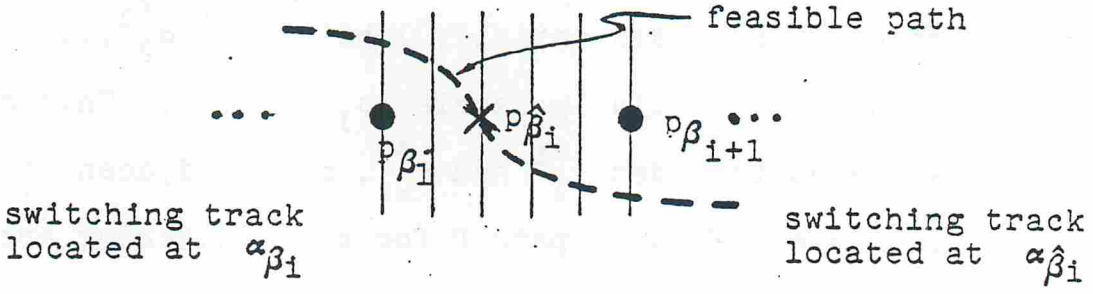


Figure 4.27. Interval $(p_{\beta_i}, p_{\beta_{i+1}})$ in D'. p_{β_i} and $p_{\hat{\beta}_i}$ are located at α_{β_i} and $\alpha_{\hat{\beta}_i}$ respectively.

After all switching arcs in P have been assigned to switching tracks, it is possible to obtain the value of all arc lengths (Δs) in G' as follows. Scan G' for an arc with an undefined Δ. Let that arc be (u,v). Then $\Delta = |\alpha_u - \alpha_v|$. Continue scanning G' until all Δs are defined.

4.5.2.5 Feasible section track assignment

In this section we describe how all feasible sections are assigned to horizontal tracks. Path P contains sequences of nodes all of which are lableed + or -. Each of these sequences is separated by a switching arc, and each sequence represents a feasible section. We can label each of these sections with an integer as shown in Table 4.3 below.

Table 4.3. Section Characteristics Required for Track Assignment and Assignment of Numeric Values to the Feasible Access Graph Arc Labels

Section	P Nodes	P Arcs	Channel Width	Inner Track
1	$\beta_{11},\beta_{12},\cdots$	$\sigma_{11},\sigma_{12},\cdots$	T_1	π_1
2	$\beta_{21},\beta_{22},\cdots$	$\sigma_{21},\sigma_{22},\cdots$	T_2	π_2
3	$\beta_{31},\beta_{32},\cdots$	$\sigma_{31},\sigma_{32},\cdots$	T_3	π_3
⋮	⋮	⋮	⋮	⋮

Associated with each section i is the width T_i of the channel in which the section lies, the number π_i of the inner occupied track nearest the section, and the set of arcs and nodes in P which represent the section.

The nodes and arcs listed in Table 4.3 are found by a simple scan of path P. The channel width T_i is the minimum ρ value in the labels of all arcs $\sigma_{i1},\sigma_{i2},\cdots$. The inner track π_i is the maximum t value in the labels of all arcs $\sigma_{i1},\sigma_{i2},\cdots$.

The actual track assignment is accomplished by taking each section i in turn and calculating the track number τ_i using the expression

$$\tau_i = \pi_i + \left\{ \frac{T_i \Omega_2 + \Omega_1}{\Omega_1 + \Omega_2} \right\} . \qquad (5)$$

The values Ω_1 and Ω_2 are calculated using Algorithm 4.3 as described earlier. The arc x used in Algorithm 4.3 for Ω_2 can be any one of the arcs $\sigma_{i1}, \sigma_{i2}, \ldots$. For Ω_1, the arc x is the section arc created in step 4 or 6 of Algorithm 4.2 as a result of feasible section i.

4.5.2.6 Updating the feasible access graph G′ to the final access graph G*

Once tracks have been assigned for N, the feasible graph G′ can be updated to become the final access graph G*. Note that G* could also be produced from the final configuration D′ (i.e., D* is configuration D with the new net N routed) directly using Algorithm 4.1 to produce the access graph. However, since G′ exists and it is faster to construct the new G* from the old G* than from D*, we do not use D*.

Algorithm 4.4: Update the feasible access graph G′ to the final access graph G*.

Purpose: This procedure replaces all literal "Σ"s (Σ is the channel capacity represented by the arc) in the arc labels of G′, and removes those arcs with zero channel capacity.

Method: This algorithm is equivalent to Algorithm 4.3 with respect to its control structure. Therefore, the function of the variables g and z^x are the same as before. The definition of T is the same as that in the previous section, namely, the track assigned to a feasible section.

Input: A feasible access graph G' derived from access graph G as per Algorithm 4.2, a path P defined on G, and the track assignmants for all sections in the path for N in D^*.

Output: An access graph G^* representing D^* where D^* is the configuration D *plus* the routed net N.

Procedure:

Step 1. Let the first node in path P be s^x. Let $g = s^x$. If s^x is adjacent to a switching arc in P, then set $z^x = s^x$ and go to step 3.

Step 2. Let z^x be the next node in P and let z^x be in section i. If z^x is the final node in P, then go to step 6.

Step 3. (a) If z^x is adjacent to a switching arc in P, then go to part (b) of this step. Otherwise, let u^x be the predecessor node of z^x in P. Let the label of (u^x, z^x) in P be (x, δ, ρ, t) and let arc (u^x, z^x) be a part of section i. Then replace the

literal "Σ" in the label of arc (u^x, z^x) in G′ with τ_i-t-1. If this value is zero, delete arc (u^x, z^x) in G′. Go to step 2.

(b) If g = 0 set $g = z^x$ and go to step 5.

Step 4. (a) If $g = s^x$, $s \neq 1$, and $g < z$ let γ be arc $(g-1, \hat{z}^x)$.

(b) If $g = s^x$, $s \neq n$, and $g > z$ let γ be arc (z^x, g).

(c) If $g \neq s^x$ and $g < z$ let γ be arc $(g, \hat{z}^x)$.

(d) If $g \neq s^x$ and $g > z$ let γ be arc $(z^x, \hat{g})$.

(e) Replace the literal "T" in the label of arc γ in G′ with τ_i. Replace the literal "Σ" in the same label with $T_i + \pi_i - \tau_i$. If this value is zero, delete arc γ from G′. Set g = 0 and go to step 5.

Step 5. (a) Let u^x be a node adjacent to z^x in G′ where $u < z$ and let the label of arc (u^x, z^x) in G be (x, δ, ρ, t). If node point P_u is within the span of a section of the recently routed path in street S^x then replace the literal "Σ" in the label of arc (u^x, z^x) in G′ with τ_i-t-1. If zero, delete arc. Otherwise, replace "Σ" with ρ.

(b) Let v^X be a node adjacent to z^X in G' where $v > z$ and let the label of arc (u^Z, v^X) in G be (x, δ, ρ, t). If v^X is the last node in P, go to step 2. Otherwise, replace "Σ" with ρ. Go to step 2.

Step 6. (a) If z^X is adjacent to a switching arc in P then go to step 7.

(b) If $g = s^X$, $s \neq 1$, $z \neq n$, and $g < z$ then let γ be arc $(g-1, z^X)$.

(c) If $g = s^X$, $s \neq n$, $z = 1$, and $g > z$ then let γ be arc (z^X-1, g).

(d) If $g \neq s^X$, $z \neq n$, and $g < z$ then let γ be arc (g, z^X).

(e) If $g \neq s^X$, $z \neq 1$, and $g > z$ then let γ be arc $(z^X-1, \hat{g})$

(f) Replace the literal "T" in the label of arc γ in G with τ_i. Replace the literal "Σ" in the same label with $T_i + \pi_i - \tau_i$. If this value is zero, delete arc γ from G'. Exit algorithm.

Step 7. (a) Replace the literal "T" in the label of arc $(z^X-1, \hat{z}^X)$ in G' with 1. Let the label of arc (z^X-1, z^X) in G be (x, δ, ρ, t).

Replace the literal "Σ" in the label of arc $(z^x-1,\hat{z}^x)$ in G′ with ρ-1.

(b) Let v^x be a node adjacent to z^x in G′ where v > z and let the label of arc (v^x,z^x) in G be (x,δ,ρ,t). If point p_v is within the span of a section of the recently routed path in street S^x then replace the literal "Σ" in the label of arc $(\hat{z}^x,v^x)$ in G′ with τ-t-1. If zero, delete arc. Exit algorithm. □

4.5.3 A comprehensive example of fixed-track routing

We now present an example of routing two nets in a configuration D_1 which already contains two routed nets. Figure 4.28 shows the original configuration. There are five horizontal tracks in each street and one vertical switching track in each interval.

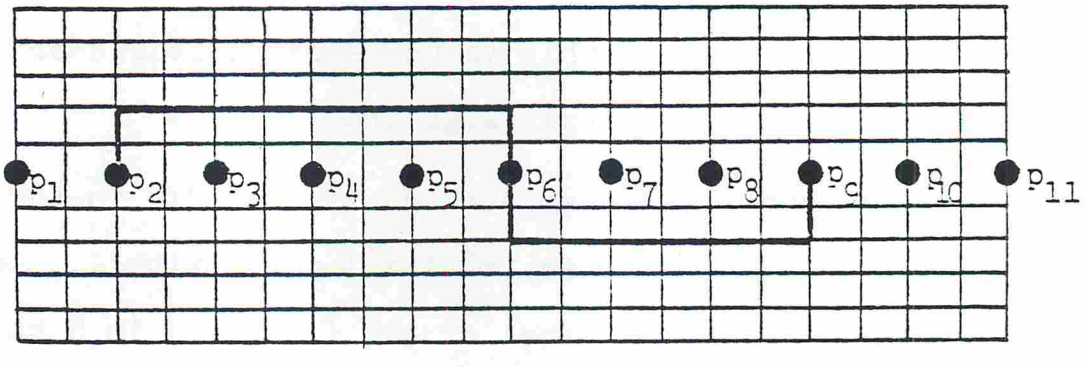


Figure 4.28. Configuration D_1 used to illustrate fixed-track routing.

Consider two nets $N_1 = (p_1, p_8)$ and $N_2 = (p_4, p_7)$. We will route N_1 first followed by N_2.

First, we obtain the access graph G_1 for D_1 using Algorithm 4.1. Graph G_1 is shown in Figure 4.29.

Next, we find a feasible minimum-congestion path P_1 for net N_1 in G_1 by using Dijkstra's algorithm as modified in Section 5.1 of this chapter. Figure 4.30 shows path P_1.

It is interesting to note that there are several minimum-congestion paths possible for N_1 in G_1. Although not to be explored in this dissertation, it is possible to find a *shortest* minimum-density path by performing a multi-variable minimum-cost path finding procedure on G_1. In fact, Dijkstra's least-cost path algorithm as given in Chapter 2 may be easily modified to do this. This subject is left for further study.

We now assign tracks in D_1 to the feasible path P_1. Since P_1 consists of one section and no switches, we will only be assigning one horizontal track to the feasible path. First we obtain the feasible access graph G_1' using Algorithm 4.2. Graph G_1 is shown in Figure 4.31. Next, we determine which horizontal track should be assigned to the single section represented by P_1. Note that the feasible section is attached to point p_1 in D'. Thus no future paths will be routed to the outside of the feasible section. We therefore assign the outermost track, track 5, to the feasible segment without having to use Algorithm 4.3 and the

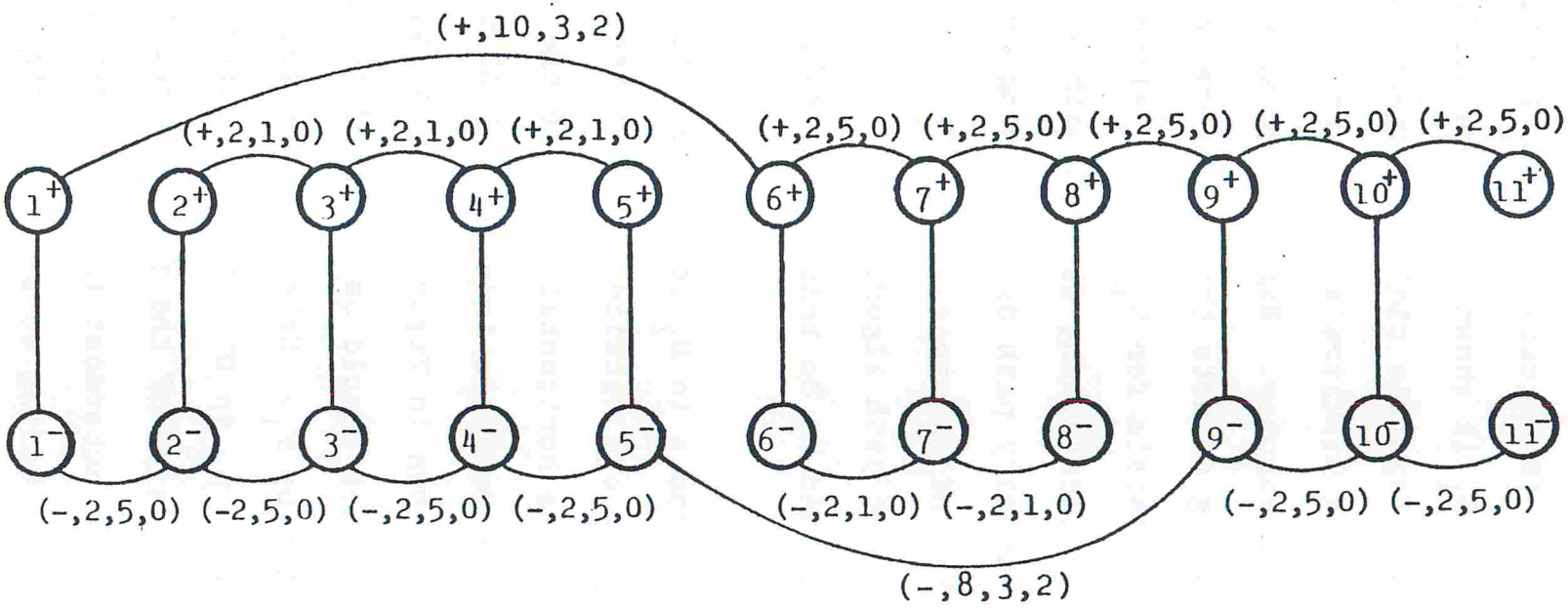


Figure 4.29. Graph G_1 representing configuration D_1.

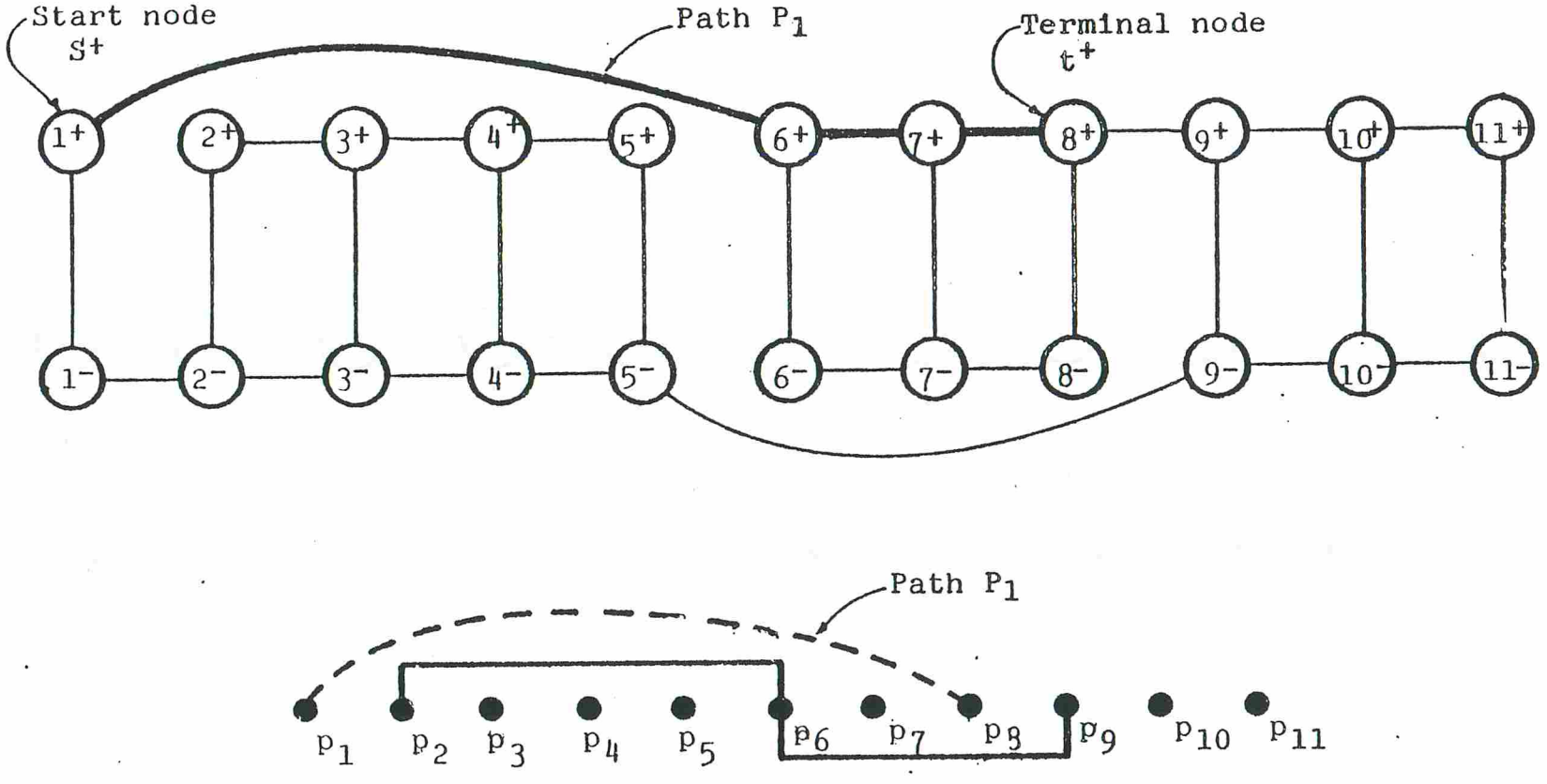


Figure 4.30. Graph G_1 and configuration D_1' showing path P_1 (reflecting a feasible minimum-congestion path in D_1 for N_1).

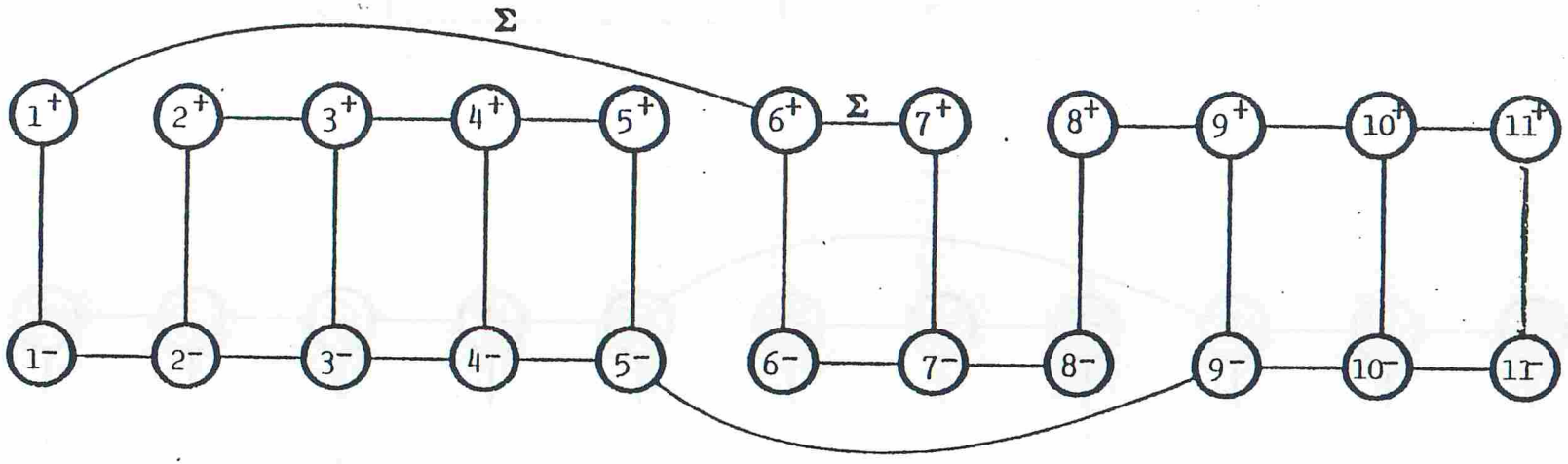


Figure 4.31. Feasible access graph G_1'. The only difference between this graph and G_1 is the substitution of the literal "Σ" for the arc channel capacity in arcs $(1^+,6^+)$ and $(6^+,7^+)$ and the deletion of arc $(7^+,8^+)$.

assignment procedure outlined in Section 4.5.2.5. The assignment procedure as outlined there would also allocate track 5 to the feasible section since $\Omega_1 = 0$ (i.e., arc x in Algorithm 4.3 does not exist), whereas $\Omega_2 = 229$ (arc x for Ω_2 can be either arc $(1^+,6^+)$ or $(6^+,7^+)$). Thus, the track T to be assigned to the feasible section is (from Eq.(5)),

$$T = \pi + \left\{ \frac{T_i \Omega_2 + \Omega_1}{\Omega_1 + \Omega_2} \right\}$$

$$= 2 + \left\{ \frac{3(229)}{229} \right\} = 5 .$$

Since the feasible path has now been assigned to a track, we have a complete characterization of the configuration D_1^* as shown in Figure 4.32.

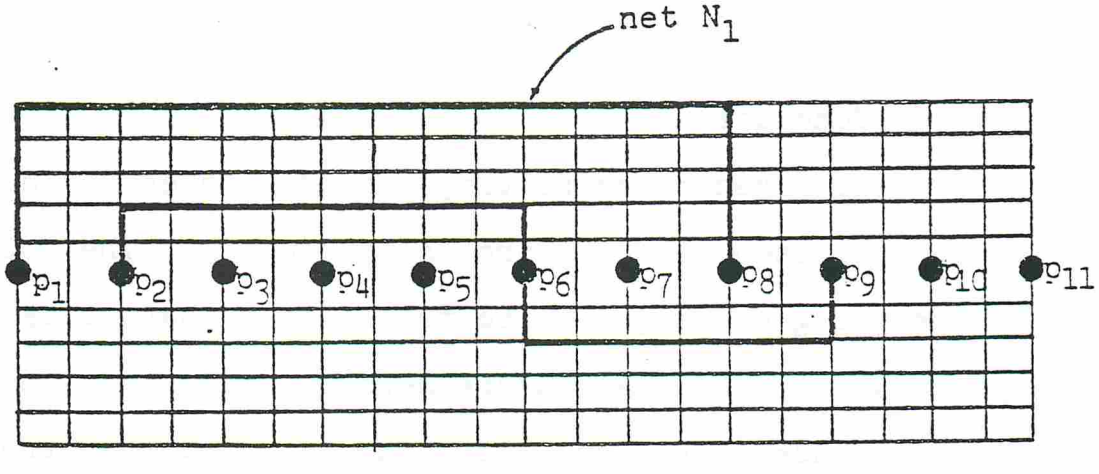


Figure 4.32. Configuration D_1^* $(= D_2)$ showing net N_1 routed in track 5 of street S^+.

Since we now want to route N_2 in D_1^*, we can update G_1' to become G_1^* by using Algorithm 4.4, or we can rename the configuration D_1^* as D_2 and obtain G_2 using Algorithm 4.1. We choose to obtain G_1^* and then rename it G_2. After processing G_1' with Algorithm 4.4, we get the graph G_1^* shown in Figure 4.33.

We now repeat the same steps for net $N_2 = (p_4, p_7)$ as we did for net N_1. We have the configuration D_2 (Fig.4.31) and the access graph G_2 (Fig.4.33). A minimum-congestion path p_2 is found in G_2. Figure 4.34 shows path P_2.

A feasible access graph G_2' for D_2' is derived from G_2 using Algorithm 4.2. G_2' is shown in Figure 4.35. There are two sections in path P_2. Table 4.4 gives the characteristics of these two sections and the tracks assigned as per Section 4.5.2.5 of this chapter. The tracks assigned to the sections are also shown in the table.

4.6 Conclusions

The fixed-track routing methods presented in this chapter represent a new approach to routing. The notion of assigning new segments to tracks such that blockage of paths for potential future nets is minimized appears to be a very useful notion. It is anticipated that the fixed-track routing method is as useful for changing the routing of existing unidirectional configurations as routing new configurations.

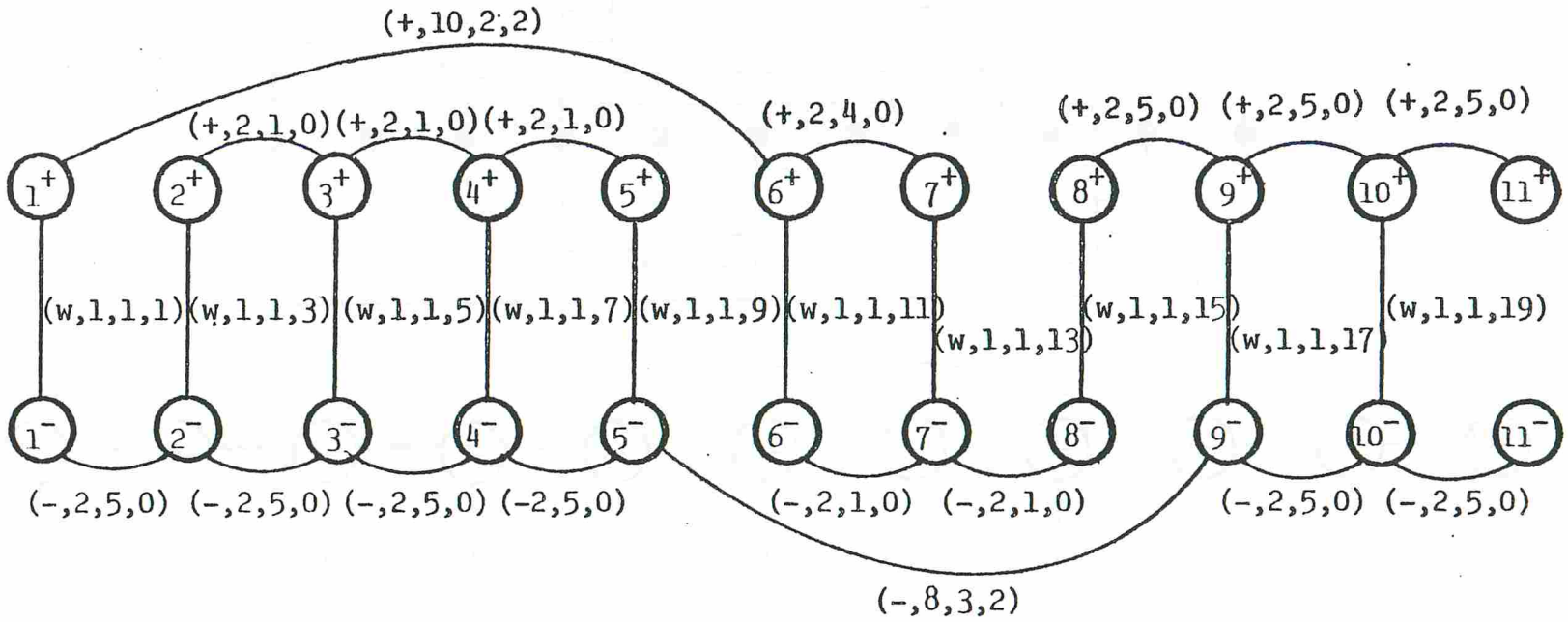


Figure 4.33. Access graph G_1^* ($=G_2$). Note that the "Σ" symbols in the arc labels of arcs $(1^+,6^+)$ and $(6^+,7^+)$ in Fig. 4.32 have been replaced with the appropriate channel widths 2 and 4 respectively.

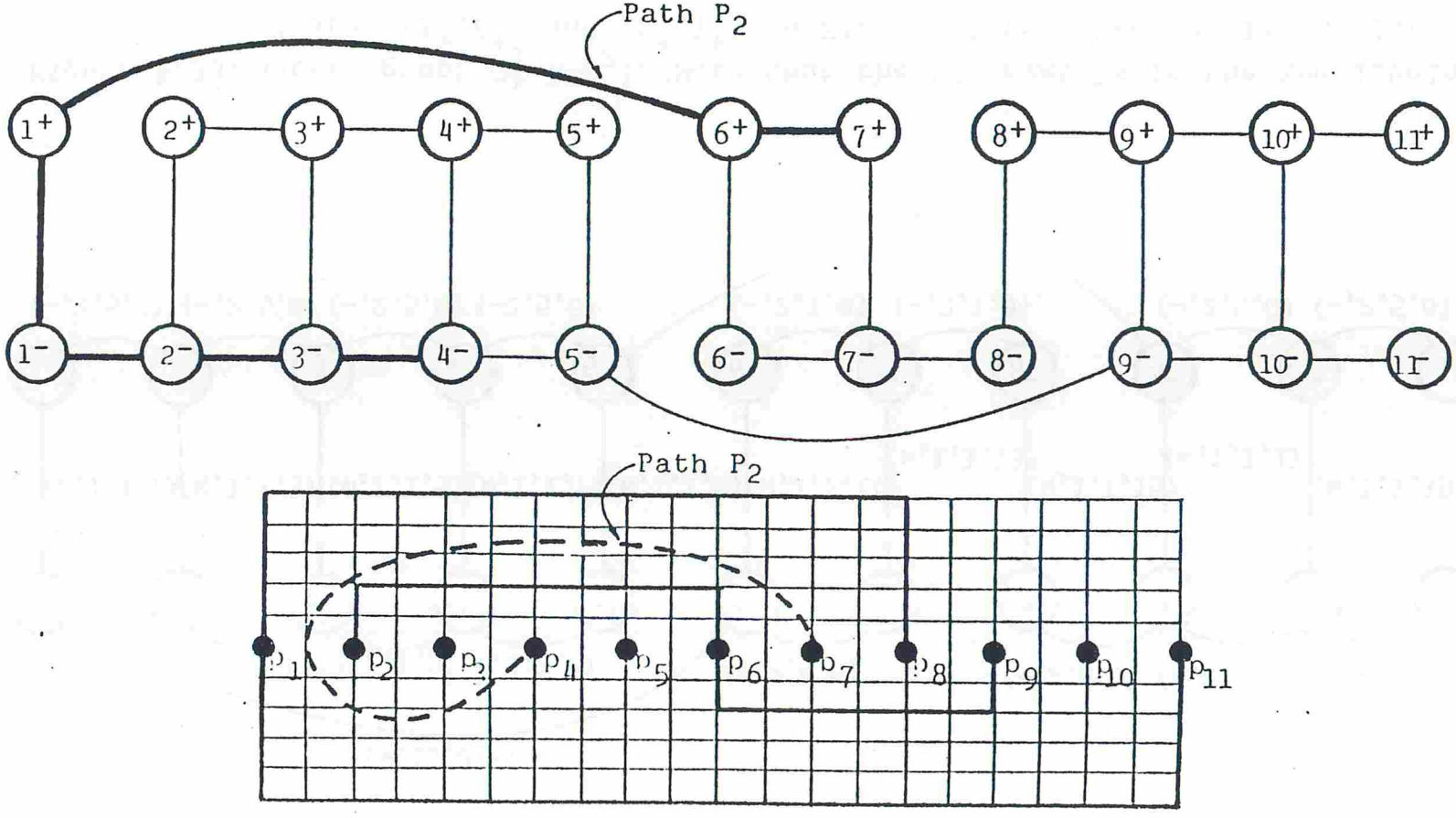


Figure 4.34. Access graph G_2 and feasible configuration D_2' showing path P_2.

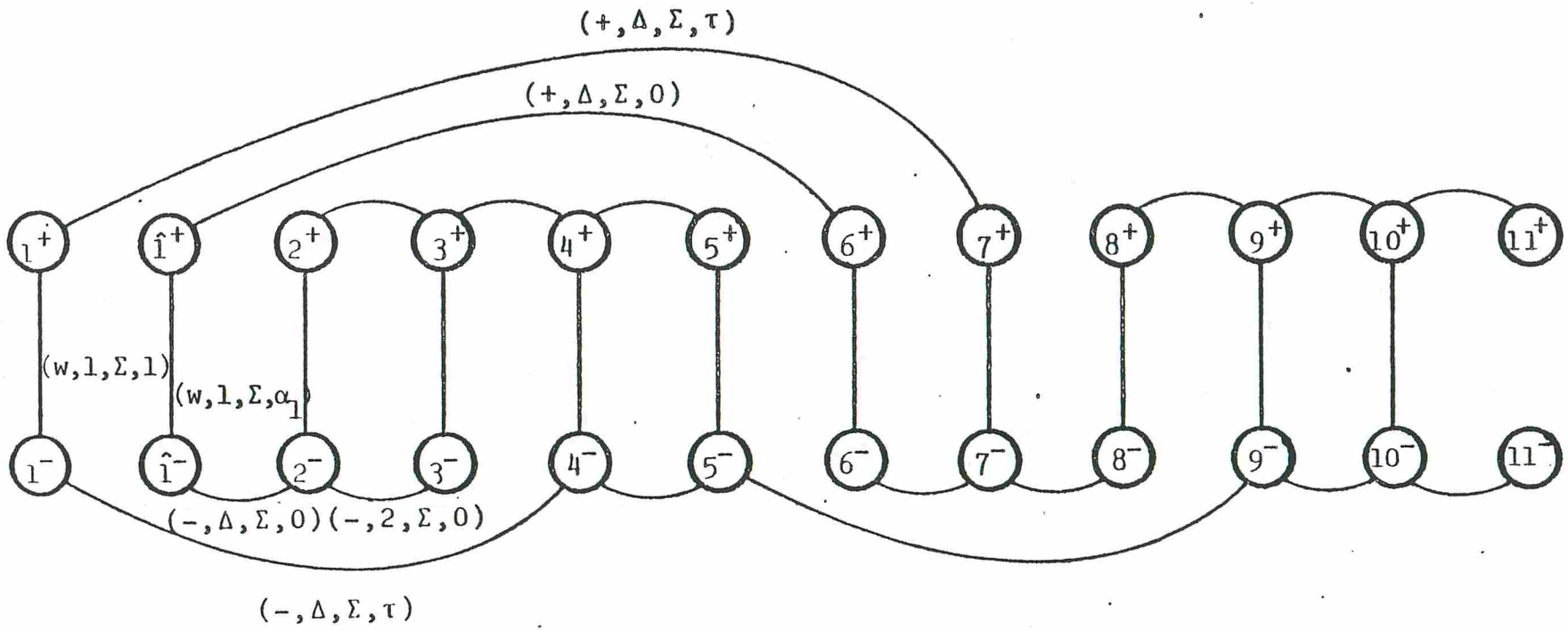


Figure 4.35. Feasible access graph G'_2. Note that the feasible switch in interval 1 in D'_2 in Fig. 4.34 has created pairs of nodes 1^+, $\hat{1}^+$, 1^-, and $\hat{1}^-$.

Table 4.4. Section characteristics and track assignments for the two section comprising the routed path for net N_2.

Section Characteristics					Track Assignment			
Section 1	Path P_2 Nodes in Section	Path P_2 Arcs in Section	D_2 Channel Width (T_1)	Inner Track (π_1)	Assigned Track (τ)	Street	Arc x in Alg.4.3	Ω
1	$4^-,3^-,2^-,1^-$	$(1^-,2^-)$, $(2^-,3^-)$, $(3^-,4^-)$	5	0	3	S^-	$\Omega_1 \rightarrow (1^-,4^-)$ $\Omega_2 \rightarrow (\hat{1}^-,2^-$ or $(2^-,3^-)$	263 263
2	$1^+,6^+$	$(1^+,6^+)$	2	2	3	S^+	$\Omega_1 \rightarrow (1^+,7^+)$ $\Omega_2 \rightarrow (\hat{1}^+,6^+)$	263 263

Although the (worst-case) complexity of fixed-track routing methods as derived in this chapter is $\mathcal{O}(n^3 \log n)$ for n-1 nets and n points, realistic configurations (even those with a large number of nets) seem to require only $\mathcal{O}(n^2)$ operations. Thus, the method given here is practical and requires little more time than in the unconstrained case. Further evidence of this may be seen in Chapter 9. Also, as in the unconstrained case described in Chapter 3, if one or more paths exist for a net in a fixed-track configuration, the method presented here guarantees that not only will a path be found, but that the chosen path is optimal for either of the goals presented; namely, minimum length or minimum congestion.

CHAPTER 5

FLOATING-TRACK UNIDIRECTIONAL ROUTING

5.0 Introduction

Whereas in the previous chapter track assignments were fixed, in this chapter sections and switches for routed paths may be reassigned to other tracks (i.e., "floated") (within minor constraints) in order to create a feasible path for a net being added to the configuration. The primary motivation for considering floating-track routing is to enhance the probability of routing completion in a unidirectional configuration. To achieve this goal a new graph model is required.

The floating-track access graph is first presented and then three routing cost goals are considered in turn, namely: minimum-routed path length, minimum-congestion, and minimum-routed path perturbation. Minimum-length and minimum-congestion routing were considered at length in Chapters 3 and 4, respectively, and are therefore only briefly treated here. Minimum perturbation routing is unique to the floating-track case.

5.1 Density

We define a *density function* $d^x(p_i)$ to be the number of routed paths intersected by an imaginary vertical line

crossing all tracks in street S^x at point location p_i, where p_i can take on values only at points $p_1, p_2, \ldots, p_n$ in a configuration D. The *density* at point p_i in street S^x is the sum of all sections in S^x intersected by the imaginary vertical line passing through p_i *including* the section covering p_i. For instance, the routed configutation shown in Figure 5.1 has the densities for both streets S^+ and S^- as shown.

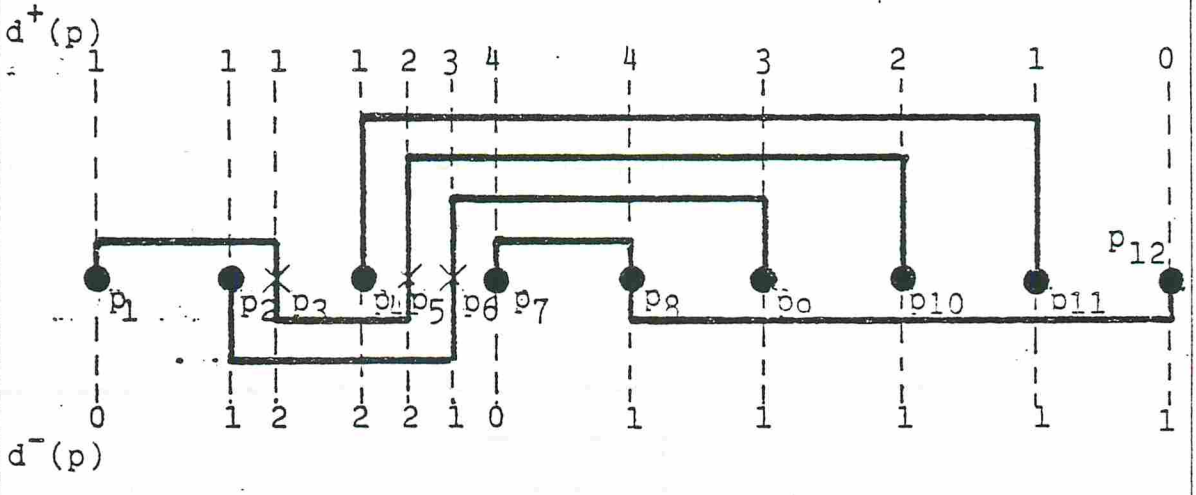


Figure 5.1. Example showing street densities.

Whereas the street densities $d^+(p_i)$ and $d^-(p_i)$ are defined for each point $p_1, p_2, \ldots, p_n$, the interval densities

$d^W(a)$ are defined only for original points. The *interval density* $d^W(a)$ is the number of pseudo-points in the unidirectional interval a. For instance, the density of interval p_4 is 2, whereas the density of interval p_3 is undefined.

The *segment density* $\rho^x(u,v)$ of a segment (p_u,p_v) is defined to be $\max_{p_u \leqq p \leqq p_v} d^x(p)$ where u and v are in C and the segment lies in street S^x. Since a segment is uniquely defined by its end points, and it lies in only one street, the superscript x is superfluous and is therefore not needed. The segment (p_1,p_3), for instance, in Figure 5.1 has segment density $\rho(1,3) = \max\{1,1,1\} = 1$. The *inner* segment density of a segment γ is the density of a segment where only sections (including γ) between the point line and the segment γ are considered. For instance, in Figure 5.2 the segment density of segment γ is 4 and the inner segment density is 3.

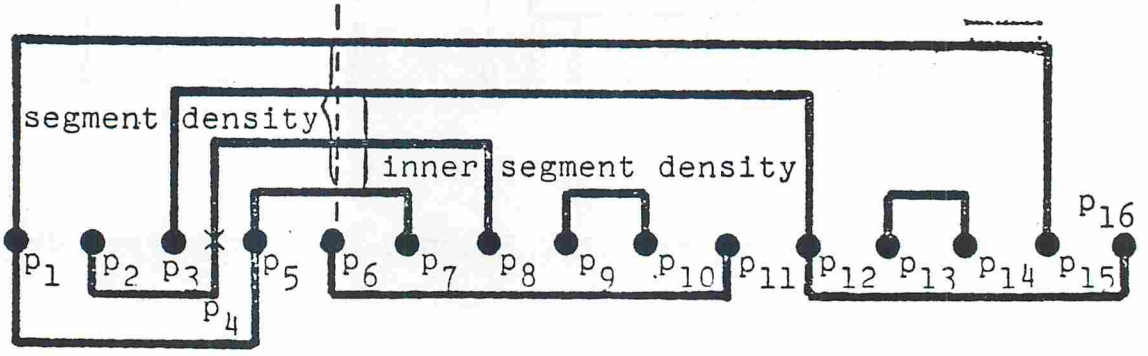


Figure 5.2. A configuration illustrating segment densities.

5.2 The Access Graph for the Floating-Track Routing Problem

As for the fixed-track access graph, the floating-track access graph is a subgraph of the unconstrained access graph. Each arc in the floating-track access graph G is labeled with a five-tuple of the form

$$(\text{type}, \delta, \rho, t, \lambda) .$$

The parameters in the label are:

type - "+", "-", or "w" depending on whether the arc is in the upper street, the lower street, or is a switch, respectively.

δ - the length of the feasible c-section represented by the arc.

ρ - the maximum street density over the length of the feasible c-section represented by the arc.

t - the track occupied by the nearest outer routed section to the feasible section represented by the arc. If there is no outer routed section, then t is the outer-most track plus one.

λ - the number of routed sections that must be moved to accommodate the feasible c-section represented by the arc.

Each of these parameters is required by the floating-track

routing algorithms developed in detail later in this study. Specifically, the length δ, density ρ, and number of perturbed sections λ are used to determine minimum-cost feasible paths. The track t and parameter λ are used to route feasible paths.

In the algorithm presented here for generating the floating-track access graph G, the depth of stack S at the time an arc is created is used to obtain ρ in the label of that arc. As the algorithm progresses, the size of stack S grows and shrinks in direct proportion to the density $d^x(p_i)$ of each point p_i in D. A record of the maximum value of $d^x(p)$ is kept for each c-segment represented in the stack. This value is ρ in the arc label. Since a list of all routed sections and their assigned tracks are a part of the description of D, the value t is directly available.

Algorithm 5.1: Construction of the floating-track access graph.

Purpose: This algorithm generates a weighted undirected graph G (called a floating-track access graph) for a given routed configuration D where street capacities are considered and track assignments for routed path may be reassigned as necessary to improve the routability of a feasible path.

Method: The point line is scanned twice from left-to-right, first for street S^+, and then for street S^-. During each

scan, successive adjacent points p_i and p_{i+1} are examined. Appropriate stack operations and graph construction are performed in steps 5-9 of the procedure based upon which of the nine adjacent point configurations (see Table 2.2, Chapter 2) is applicable. Steps 5-10 are performed only if tracks are available for the feasible segment being processed. Let ψ be a channel whose left end is at interval u_j. Then the stack contains elements of the form (j,ρ,t,λ) where ρ represents the maximum segment density of a feasible segment routed in ψ, t is the track occupied by the outer routed section bounding the channel, and λ is the number of sections which must be moved to permit a feasible c-section to be routed in ψ. As for Algorithm 3.1, we define point p_{n+1} to be an uncovered imaginary point lying just to the right of point p_n on the point line in D.

Input: Routed configuration D.

Output: The floating track access graph G for D.

Procedure:

Step 1. (initialization) Set x = "+", and clear stack S. Construct the 2n nodes $1^+,2^+,\ldots,n^+$, $1^-,2^-,\ldots,n^-$.

Step 2. (switching arcs) Consider each point p_i, $i = 1,2,\ldots,n$. If point p_i is a unidirectional point and the number ϕ of pseudo-points in the unidirectional interval with p_i as its left

point is less than the switching track capacity T^W then construct an arc between nodes i^+ and i^- and label it $(w,1,\phi,\alpha_i,0)$. If point p_i is a pseudo-point, let it be in unidirectional interval u. If the number ϕ of pseudo-points (including p_i) in u is less than the switching track capacity T^W then construct an arc between nodes i^+ and i^- and label it $(w,1,\phi,\alpha_i,0)$.

Step 3. Set i = 0. Push $(0,0,T^X+1,0)$ on top of stack S.

Step 4. (main loop) Set i = i+1. Let the top item in stack S be (j,ρ,t,λ). Consider points p_i and p_{i+1} in D. Select the applicable case from Table 5.1 and branch to the step indicated. (In steps 5 through 7, let t_i be the track occupied by the routed segment whose end-point is i. If no such routed segment exists, set $t_i = 0$.)

Step 5. Pop S to get (j,ρ,t,λ). If $t < T^X$ and $j \neq 0$, then construct an arc (i,j) in G and label it $(x,\alpha_i-\alpha_j,\rho,t,\beta)$. Set $\beta = \lambda$ if $t-t_i = 1$. Otherwise set $\beta = 0$. Push (i, depth (S),t,λ) on S regardless of the value of j or t. Go to step 9.

Step 6. Let (j,ρ,t,λ) be the top element in stack S. Push $(i,0,t_i,\beta)$ on S and for every element

in S set the second field to itself or depth (S)-1 whichever is greater. Set $\beta = \lambda+1$ if $t-t_i = 1$. Otherwise set $\beta = 1$. Go to step 9.

Step 7. Pop S to get (j,ρ,t,λ). If $t < T^x$ and $j \neq 0$ then construct an arc (i,j) and label it $(x, \alpha_i-\alpha_j, \rho, t, \beta)$. Set $\beta = \lambda$ if $t-t_i = 1$. Otherwise set $\beta = 0$. Go to step 9.

Step 8. For every element in stack S, set the second field either to itself or to the depth of S, whichever is greater.

Step 9. (end of main loop) If $i \neq n$, then go to step 4.

Step 10. If x = "+", then set x = "-", set stack S empty, and go to step 3.

5.2.1 Example of operation of Algorithm 5.1

As an example of the operation of Algorithm 5.1, consider again the configuration shown in Figure 4.4 (Chapter 4) which is repeated in Figure 5.3 for convenience. For this example, $T^+ = 4$, $T^- = 2$, and $T^W = 1$.

Algorithm 5.1 begins by creating 20 nodes and 7 switching arc each labeled (w,1,1,0,0) as directed by steps 1, 2, and 3. Stack S has one element as shown in Figure 5.4(a). Table 5.2 shows the oepration of this algorithm for street S^+.

Table 5.1. The Jump Table for Algorithm 5.1.

Adjacent Point Configuration (Table 2.2)	IF: p_i is Covered in S^x	IF: p_{i+1} is Covered in S^x	IF: p_i is Left Point in Segment	IF: p_{i+1} is Left Point in Segment	THEN: Go To Step
1	no	no	—	—	5
2	yes	no	no	—	5
3	no	yes	—	yes	5
4	no	yes	—	no	7
5	yes	yes	no	no	7
6	yes	yes	no	yes	5
7	yes	no	yes	—	6
8	yes	yes	yes	yes	6
9	yes	yes	yes	no	8

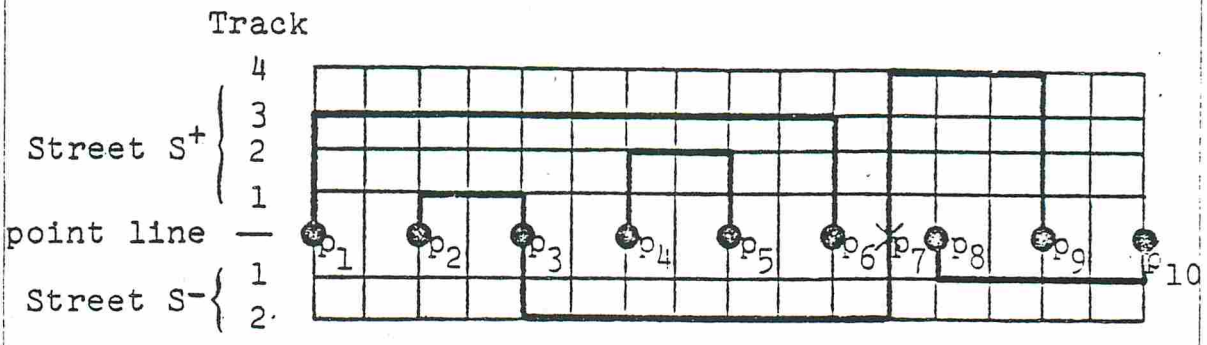


Figure 5.3. A configuration illustrating the use of Algorithm 5.1

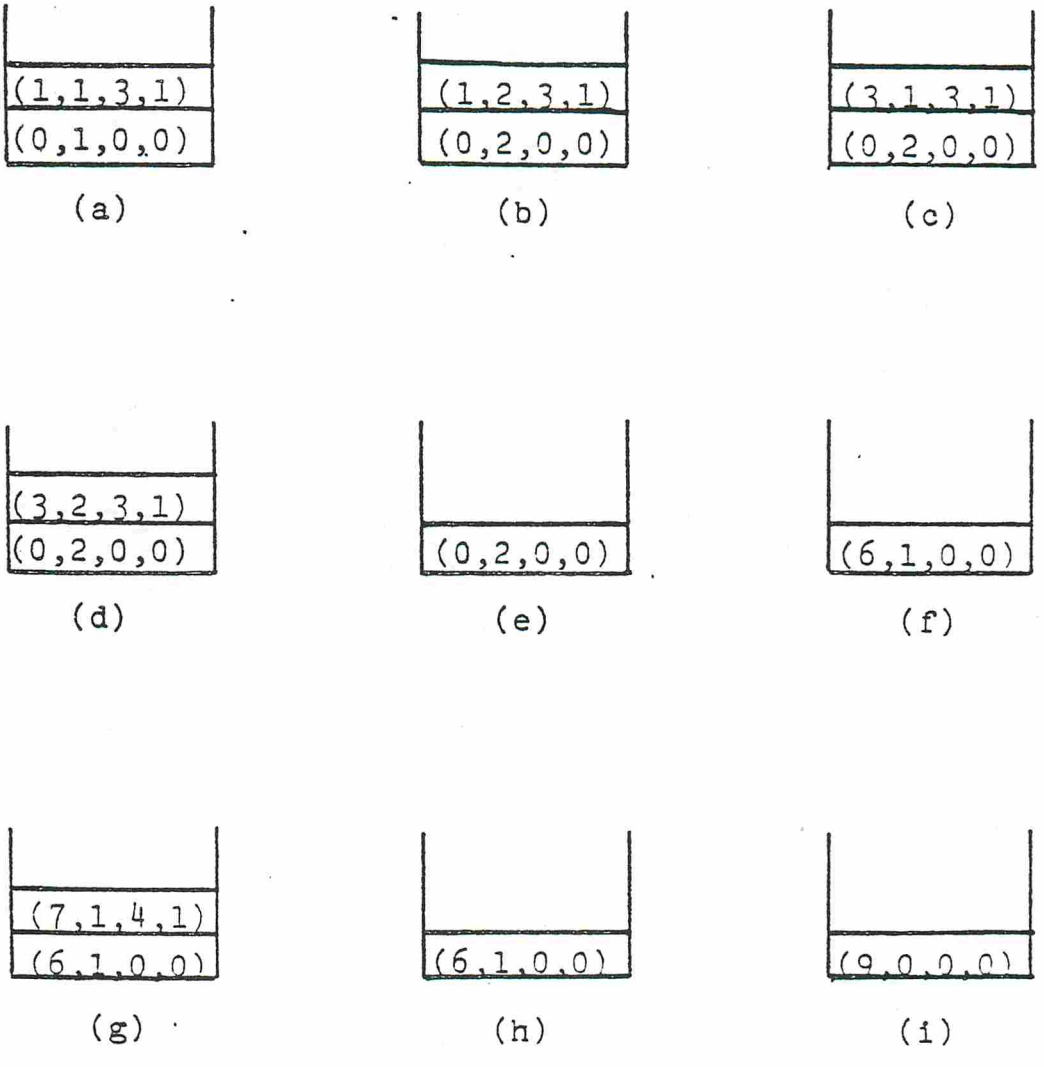


Figure 5.4. Stack contents for example of Algorithm 5.1.

Table 5.2. Algorithm 5.1 processing steps for example. Only the operations performed for street S^+ are shown.

i	Process Step	Stack Action	Fig.5.4	Arc	Arc Label
1	6	Push (1,0,3,1). For each element in S, set the second field to 1	(a)		
2	8	For each element in S, set the second field to itself or 2 whichever is greater	(b)		
3	5	Pop (1,2,3,1). Push (3,1,3,1).	(c)	(1,3)	(+,4,2,3,0)
4	8	For each element in S, set the second field to itself or 2 whichever is greater	(d)		
5	7	Pop (3,2,3,1).	(e)	(3,5)	(+,4,2,3,1)
6	5	Pop (0,2,0,0). Push (6,1,0,0).	(f)		
7	6	Push (7,0,4,1). For each element in S, set the second field to itself or 1 whichever is greater.	(g)		
8	7	Pop (7,1,4,1)	(h)	(7,8)	(+,1,1,4,0)
9	5	Pop (6,1,0,0). Push (9,0,0,0).	(i)		
10	5	Pop (9,0,0,0). Push 10,0,0,0).	-	(9,10)	(+,2,0,0,0)

The graph resulting from processing the configuration D in Figure 5.3 with Algorithm 5.1 is shown in Figure 5.5. Note that the graph structure is similar to that in Figure 4.6 (Chapter 4) since this example is essentially unconstrained (i.e., all channels contain at least one free track) with the exception of switching paths $(6^+,6^-)$ and $(7^+,7^-)$.

5.2.2 Execution time of Algorithm 5.1

The control structure of Algoirthm 5.1 is equivalent to that of Algorithm 3.1. The analysis given in Section 3.3.2 of Chapter 3 is thus applicable to this case except that the number of operations per step differs. Table 5.3 gives the number of operations in each step of Algorithm 5.1. d is the depth of stack S.

Table 5.3. Number of operations per step in Algorithm 5.1 as a function of the number of unidirectional points n.

Step	# of Operations	# of Times Each Step is Entered
1	2n+2	1
2	3n-3	1
3	2	2
4	2	2n
5	8	see discussion
6	4+3d	see discussion
7	7	see discussion
8	2d	see discussion
9	1	2n
10	3	2

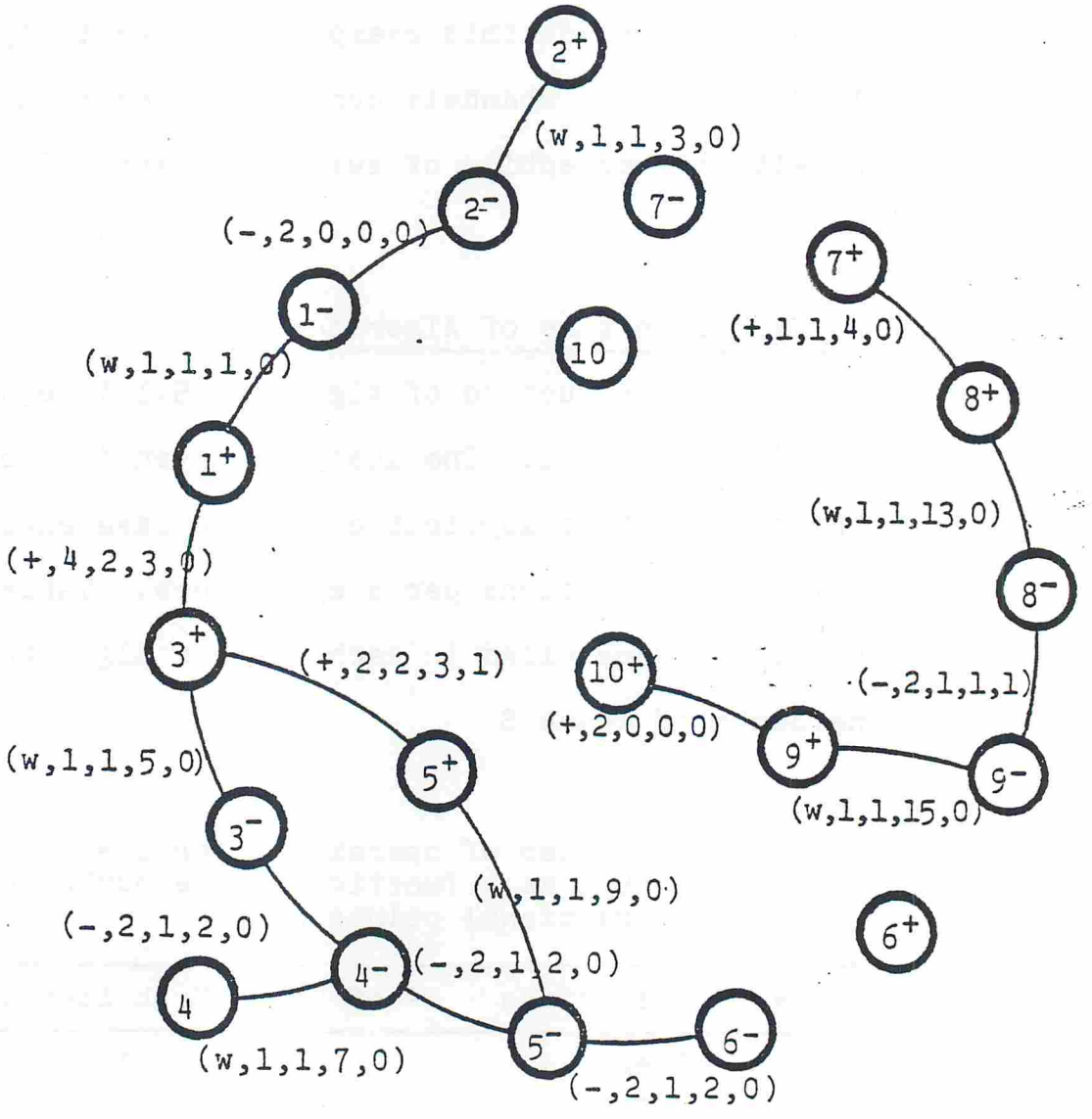


Figure 5.5. Access graph for configuration D shown in Fig.5.3. This graph was generated by Algorithm 5.1

The precedence graph G_p for the floating-track access graph construction algorithm is shown in Figure 5.6. The only difference between this graph and the graph G_p in Chapter 3 (Fig.3.4) is the number of operations required by each of the steps (shown as underlined integers adjacent to each node in Fig. 3.4 and Fig. 5.6).

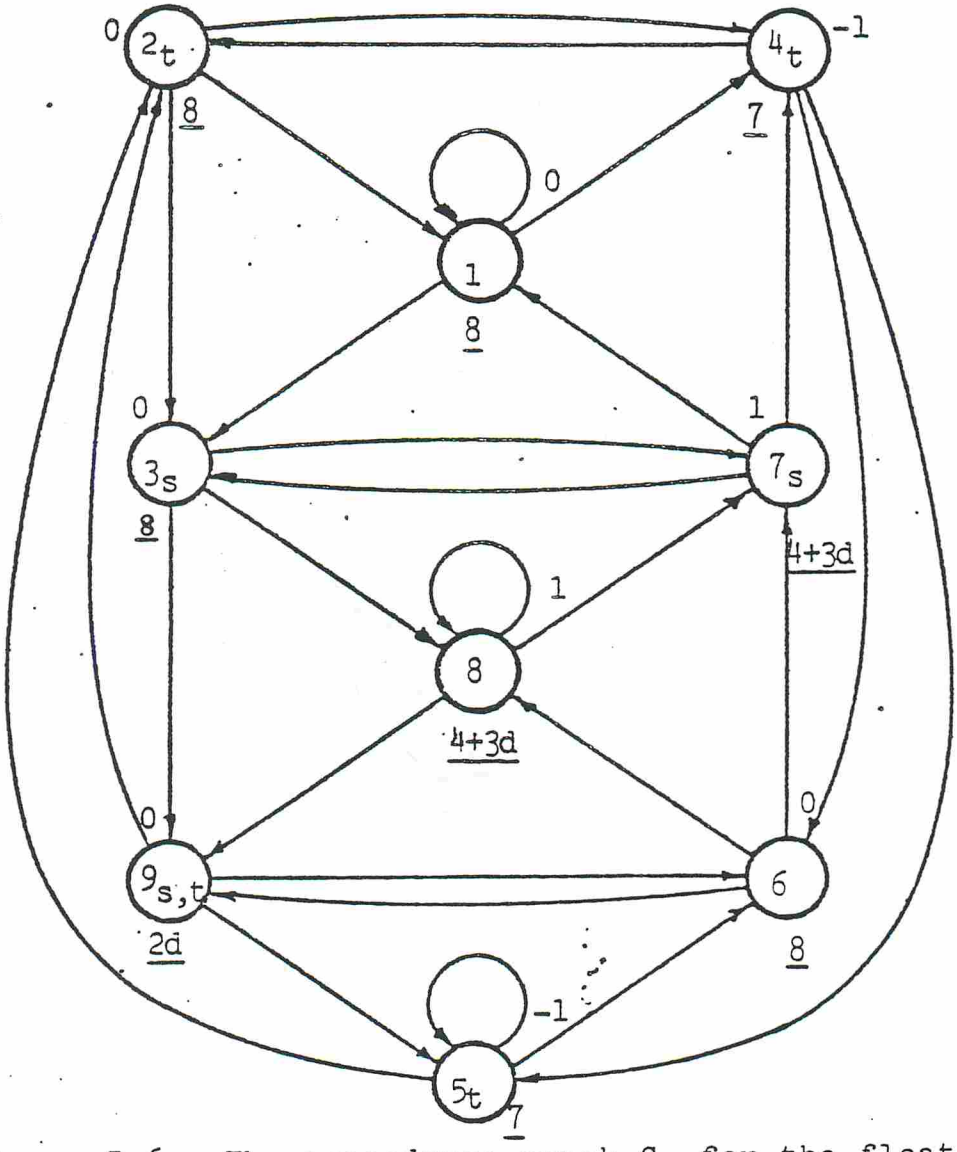


Figure 5.6. The precedence graph G_p for the floating-track access graph construction algorithm, Algorithm 5.1. See Section 3.3.2, Chapter 3, for a complete explanation of this graph.

Graph G_p can be collapsed to G'_p as was done in Chapter 3 using the same set definitions as shown there, namely:

node A in G'_p is {1,2,3,6},

node B in G'_p is {7,8},

node C in G'_p is {4,5},

node D in G'_p is {9},

where the elements of those are nodes in G_p. The graph G'_p is shown in Figure 5.7.

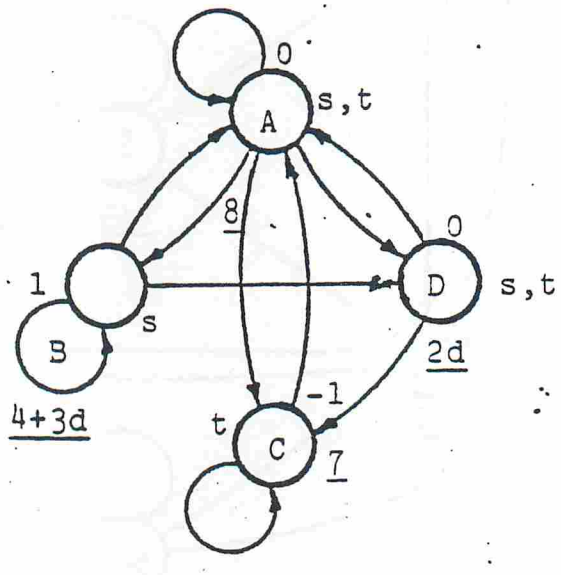


Figure 5.7. The simplified precedence graph G'_p.

We need one more piece of information before the maximum number of operations required by Algorithm 5.1 can be ascertained. In Algorithm 5.1, when point p_i is being processed the depth d of stack S equals $d^x(p_i)$. Note that for $d \geq 2$, the weight of node B is greater than that of any other node in G'_p. If we maximize the number of times node B appears in a path P in G'_p, the total number of operations required by Algorithm 5.1 processing a configuration reflected by that path will be maximized. A valid path in G'_p which accomplishes this is

$$B^k \; D \; C^k .$$

In terms of graph G_p, the path is

$$8^k \; 9 \; 5^k .$$

The configuration reflecting this sequence is shown in Figure 5.8.

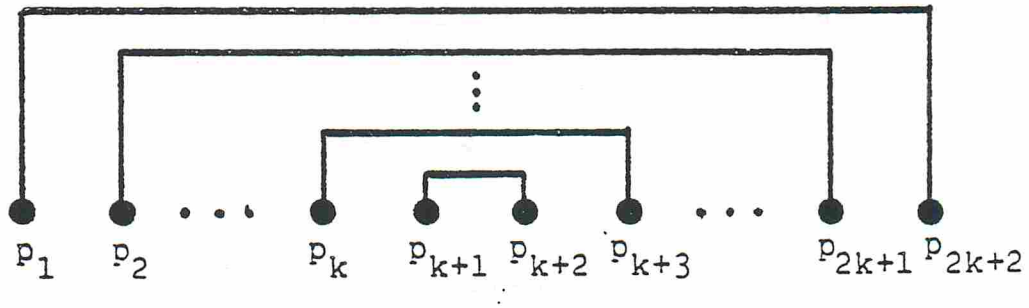


Figure 5.8. A configuration requiring the maximum number of operations by Algorithm 5.1.

Note that for the configuration shown in Figure 5.8, the depth of stack S grows from 1, when the algorithm begins, to k when the algorithm is processing interval u_k. Since step 6 has been repeatedly reentered during this time, the number of operations $\mathcal{O}ps_k$ performed by Algorithm 5.1 up to and including interval u_k is

$$\mathcal{O}ps_k = \underset{\text{step 1}}{(2k+2)} + \underset{\text{step 2}}{(3k-3)} + \underset{\text{step 3}}{2} + \underset{\text{step 4}}{2k} + \underset{k \text{ x step 6}}{\sum_{\ell=1}^{k} (4+3\ell)}$$

$$= 9k + 2k + \frac{k(k+1)}{2}$$

$$= \frac{1}{2}(3k^2 + 25k + 2). \qquad (1)$$

Completing the scan, and noting that $k = \frac{n}{2} - 1$, we see that

$$\mathcal{O}ps_n = \underset{\text{steps 1-4}}{(7n+1)} + \underset{k \text{ x step 6}}{\sum_{\ell=1}^{k} (4+3\ell)} + \underset{\text{step 8}}{2(k+1)} + \underset{k \text{ x step 7}}{\sum_{\ell=1}^{k} 7}$$

$$= \frac{3n^2}{8} + \frac{61}{n}n - 11$$

steps are required for one scan of the points in the configuration. For both streets, the maximum number of oper-

ations required, ∂ps_{max}, is

$$\partial ps_{max} = \frac{3}{4}n^2 + \frac{41}{2}n - 21 .$$

Thus, in the worst case, Algorithm 5.1 is $\partial(n^2)$.

Naturally, the street capacities, T^x, have an impact on the maximum depth of stack S. Thus, if T^x is small compared to the number of unidirectional points in the configuration, the maximum number of operations required by Algorithms 5.1 in the worst case is $\partial(nT^x) \approx \partial(n)$. Since for real problems, $T^x << n$, the expected average complexity for Algorithm is $\partial(n)$.

The least number of operations required by Algorithm 5.1 is for the configuration shown in Figure 5.9. Algorithm 5.1 executes ∂ps_{min} operations for this configuration where

$$\partial ps_{min} = 19n - 13 .$$

Therefore, in the best case, Algorithm 5.1 is $\partial(n)$.

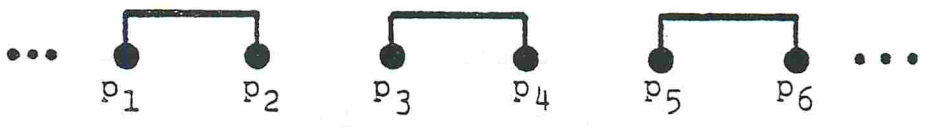


Figure 5.9. A configuration requiring the fewest number of operations by Algorithm 5.1.

5.2.3 Bounds on the size of the floating-track access graph

By Lemma 5.1 below we see that the floating-track access graph is a topological subgraph of the unconstrained access graph. Thus, Theorem 4.1 in Chapter 4 also holds for the floating-track access graph. If n is the number of unidirectional points in a configuration D and c is the number of segments routed in D, then the number of arcs k in the floating-track access graph G for D is

$$k \leq 3n-c-3 .$$

Lemma 5.1: Let G_u be an access graph for an unconstrained configuration D_u, and let G_h be an access graph for a fixed-track configuration D_h. If D_u is route equivalent to D_h, then G_h is a topological subgraph of G_u.

Proof: Similar to proof for Lemma 4.1, Chapter 4. □

5.3 Minimum-Length Paths in a Floating-Track Configuration

A minimum-length feasible path may be found in a floating-track configuration in the identical way as for the unconstrained case discussed in Chapter 3, Section 3.5. Naturally, the floating-track access graph is used in Algorithm 3.2 rather than the unconstrained access graph. The

second parameter (δ) in each arc label is used to compute the cost of the minimum-length path in the graph.

5.4 Minimum-Congestion Paths in a Floating-Track Configuration

For the floating-track model, congestion is defined in the same way as for the fixed-track case, except that here we attempt to minimize the number of wires routed in a given area rather than maximize the number of free tracks as we did in the fixed-track case. Therefore, whereas ρ was defined to be the number of free tracks in a section channel in Chapter 4, in this chapter we redefine ρ to be the segment density as defined in Section 5.1.

The same algorithm as outlined in Section 4.6.1 of Chapter 4 is used here except that $w(u_i,v) = \rho$ rather than $T^x\text{-}\phi(u_i,v)$ as in the fixed-track case.

5.5 Minimum-Perturbation Paths in a Floating-Track Configuration

In this section we consider the problem of finding a path for a net N such that a minimum number of previously routed sections need to be reassigned to new tracks in order to accommodate the assignment of tracks for the path. For instance, in the configuration shown in Figure 5.10, the minimum-perturbation path requires only one section to be moved. Assume section (12,14) is assigned to track 2 in street S^-. Note that both the minimum-congestion path and the minimum-length path require two sections to be moved.

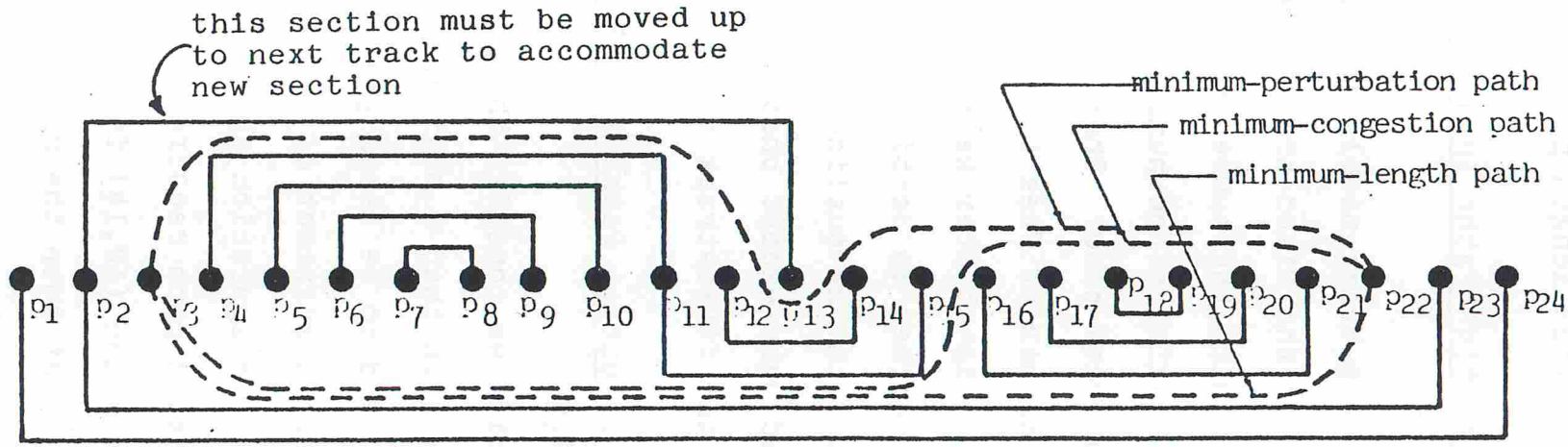


Figure 5.10. A routed configuration D illustrating the three kinds of paths discussed in this chapter.

A restriction imposed on the results presented here is that two or more sections within the same path may not overlap each other in the same street. If they do, the results of these algorithms are unpredictable, especially as the configuration becomes densely routed. Figure 5.11 shows a path where sections 1 and 3 overlap each other. If there is only one free track in street S^+ then either segment 1 or 3 cannot be assigned a track. The algorithms presented here do not account for those overlapping sections and thus any implementation must either test for and discard such paths, or make some provision for them.

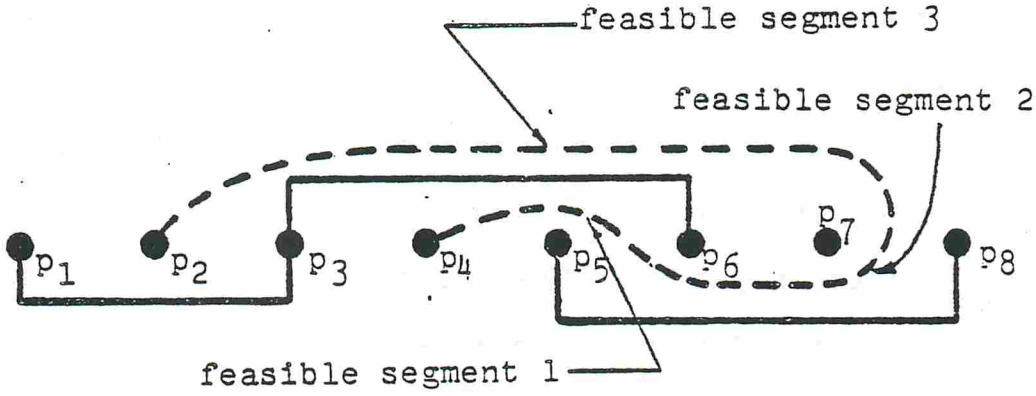


Figure 5.11. A path with overlapping segments 1 and 3 in street S^+.

Another restriction imposed by the results presented here is that only outer routed sections are moved if ne-

cessary. In order to minimize any possible adverse effect due to this restriction, we always route a new segment in the innermost available track. It thus follows that when previously routed sections must be moved to create a free track for a feasible section, it is only necessary to move outer routed sections.

5.5.1 Finding the floating-track minimum-perturbation path

Given a configuration D, a floating-track access graph G for D, and a net $N = (p_a, p_b)$ to route, we desire to find a path P between the nodes a^x, b^x in G such that some function of the arc parameter λ along P is minimum for all paths between a^x and b^x. Since all section arcs in G represent canonical channels in D, whereas P may consist of sequences of adjacent canonical channels, it is not possible to consider the optimizing function to be simply the sum of all the λs in path P. The following procedure is a modification of Dijkstra's minimum-cost path procedure which properly uses λ to obtain minimum-perturbation paths.

Algorithm 5.2: Dijkstra's minimum-cost path procedure modified to find minimum-perturbation paths.

Purpose: This procedure finds minimum-perturbation paths in a floating-track access graph from a given source node u_0 to a given terminal node x.

Method: Let S_i be a proper subset of V, the node set of G, such that $u_0 \in S_i$, and let $\bar{S}_i$ be the subset of V which consists of all nodes not in S. Let V contain η nodes. The basic idea is to construct an increasing sequence $S_0, S_1, \ldots, S_{\eta-1}$ of subsets of V in such a way that, at the end of stage i, minimum-perturbation paths from u_0 to all nodes in S_i are known. We define a label $\ell(u_i)$ for each node u_i where $\ell(u_i)$ will be the number of routed segments perturbed by routing the feasible path represented by the minimum-perturbation path from u_0 to u_i when the algorithm has processed set S_i. We further define a node label $\mu(u_i)$ which is used by the algorithm to keep a correct record of the number of perturbed sections required by the minimum-perturbation path containing node u_i.

Input: A floating-track access graph G, a source node u_0, and a terminal node x.

Output: The minimum-perturbation path in G between u_0 and x. PRED(u_i) points to the next node adjacent to u_i in the path beginning at x and ending at u_0.

Procedure:

Step 1. Let $\ell(u_0) = 0$, PRED(u_0) = u_0, $\ell(v) = \infty$ for all $v \in G$ and $v \neq u_0$, and $S_0 = \{u_0\}$.

Step 2. For each node v in $\bar{S}_i$:

(a) if arc $(u_i, PRED(u_i))$ is not the same type as arc (u_i, v) go to part (b) of this step. If $\lambda(u_i, PRED(u_i)) \geq \lambda(u_i, v)$ and if $\ell(u_i) < \ell(v)$ then set $PRED(v) = u_i$ and replace $\ell(v)$ by $\ell(u_i)$. If $\lambda(u_i, PRED(u_i)) < \lambda(u_i, v)$ and if $\mu(u_i) + \lambda(u_i, v) < \ell(v)$ then set $PRED(v) = u_i$ and replace $\ell(v)$ by $\mu(u_i) + \lambda(u_i, v)$. Go to part (c) of this step.

(b) if $\ell(u_i) + \lambda(u_i v) < \ell(v)$ set $PRED(v) = u_i$, $\mu(v) = \ell(u_i)$, and replace $\ell(v)$ by $\ell(u_i) + \lambda(u_i, v)$.

(c) compute $\min_{v \in \bar{S}} \{\ell(v)\}$ and let u_{i+1} denote a node v for which the minimum is attained. Set $S_{i+1} = S_i \cup \{u_{i+1}\}$.

Step 3. If $i = \eta - 1$, exit procedure since no path exists between u_0 and x. If $u_{i+1} \neq x$, replace i by i+1 and go to step 2. If $u_{i+1} = x$, exit procedure since the minimum-perturbation path has been found between u_0 and x. □

This algorithm executed the same total number of steps as Algorithm 2.1, namely (for n nodes in G)

$$6n+(2n+1)\log(2n-1)+\log(n)+3\sum_{i=1}^{n-1}\log(2n-i-1) \text{ operations .}$$

5.5.2 An example of the use of Algorithm 5.2

Consider the configuration shown in Figure 5.12 and assume net N = (1,13) is to be routed. The floating-track access graph for the configuration is shown in Figure 5.13.

Algorithm 5.2 begins by setting $\ell(1^+) = 0$, $PRED(1^+) = 1^+$, $\ell(v) = \infty$ for all $v \in G$ and $v \neq 1^+$, and $S_0 = \{1^+\}$. A record of the execution of Algorithm 5.2 for this example is shown in Table 5.4. Each row represents one complete iteration of step 2. Note from this table, the number of sections that must be moved to accommodate feasible path (1,13) is 2.

5.5.3 Track assignment for minimum-perturbation routing

Algorithm 5.3: Minimum-perturbation path track assignment.

Purpose: This procedure moves paths to adjacent tracks as necessary and assigns the feasible path as found by Algorithm 5.2 to tracks in the configuration D.

Method: This simple procedure sequentially scans the minimum-perturbation feasible path in D, moving routed sections one track further away from the point line as necessary and routing the feasible sections. Track assignments for switches are not considered.

Input: A routed configuration D and a feasible minimum-perturbation path P for D.

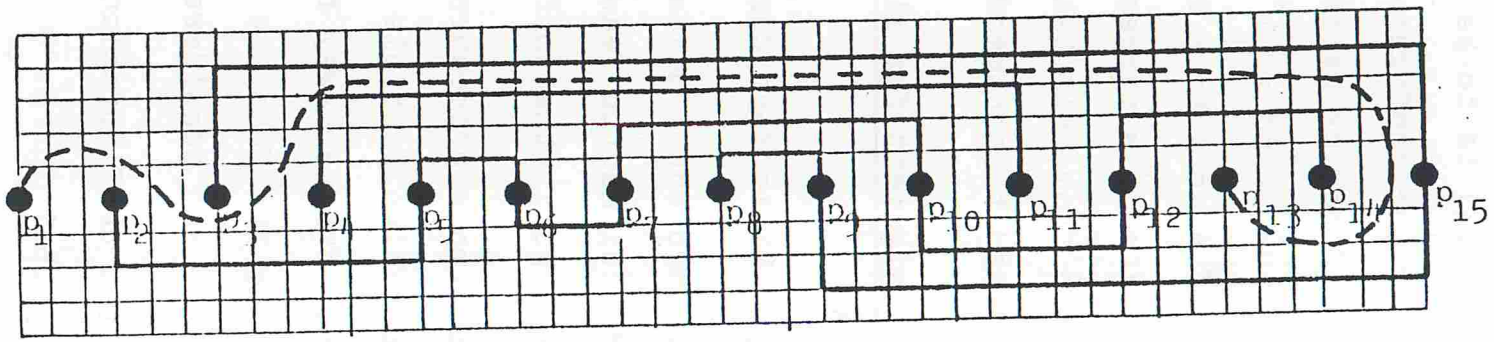


Figure 5.12. A configuration illustrating the use of Algorithm 5.2.

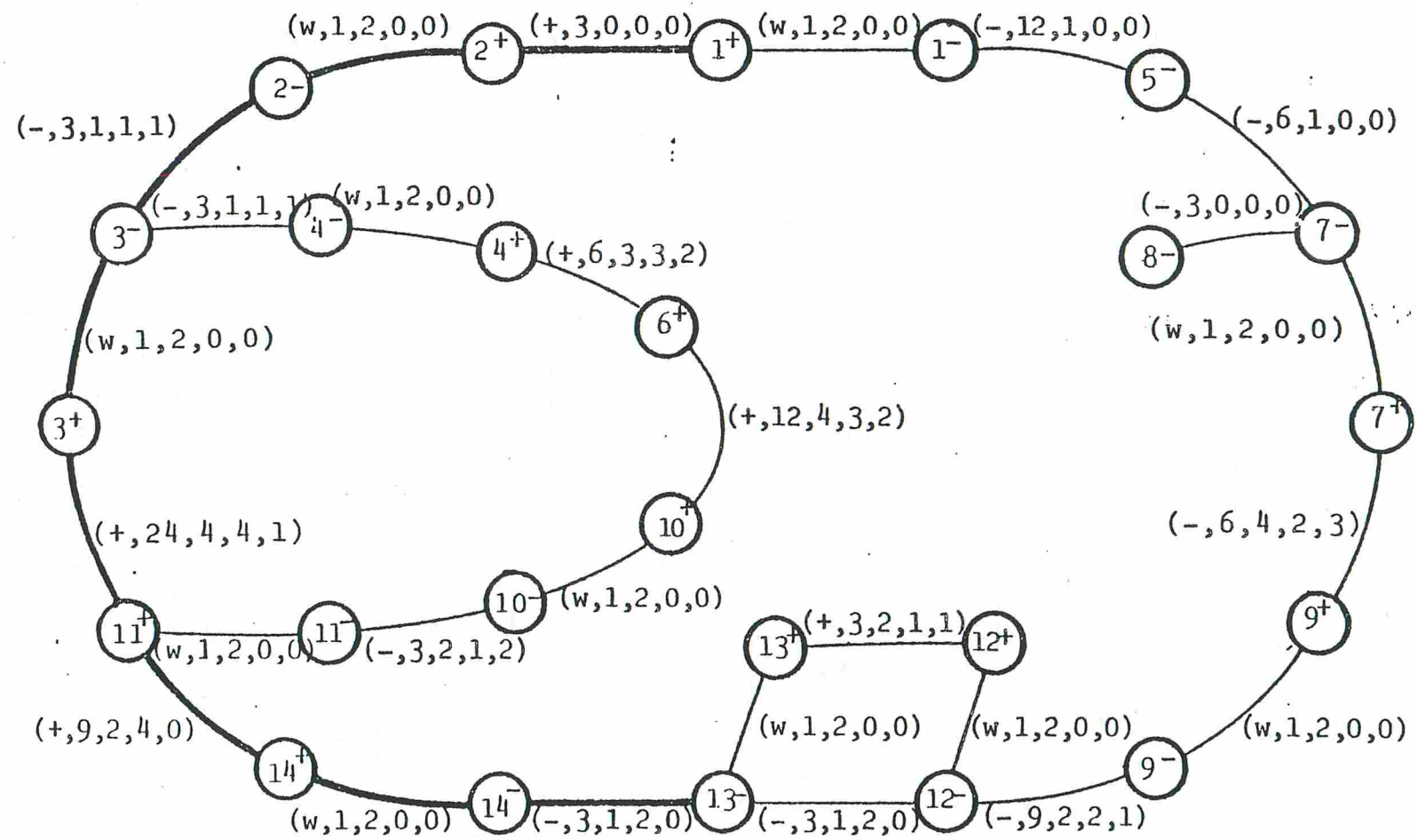


Figure 5.13. The floating-track access graph for the configuration shown in Fig.5.12. Isolated nodes 5^+,6^-,8^+,15^+, and 15^- are not shown.

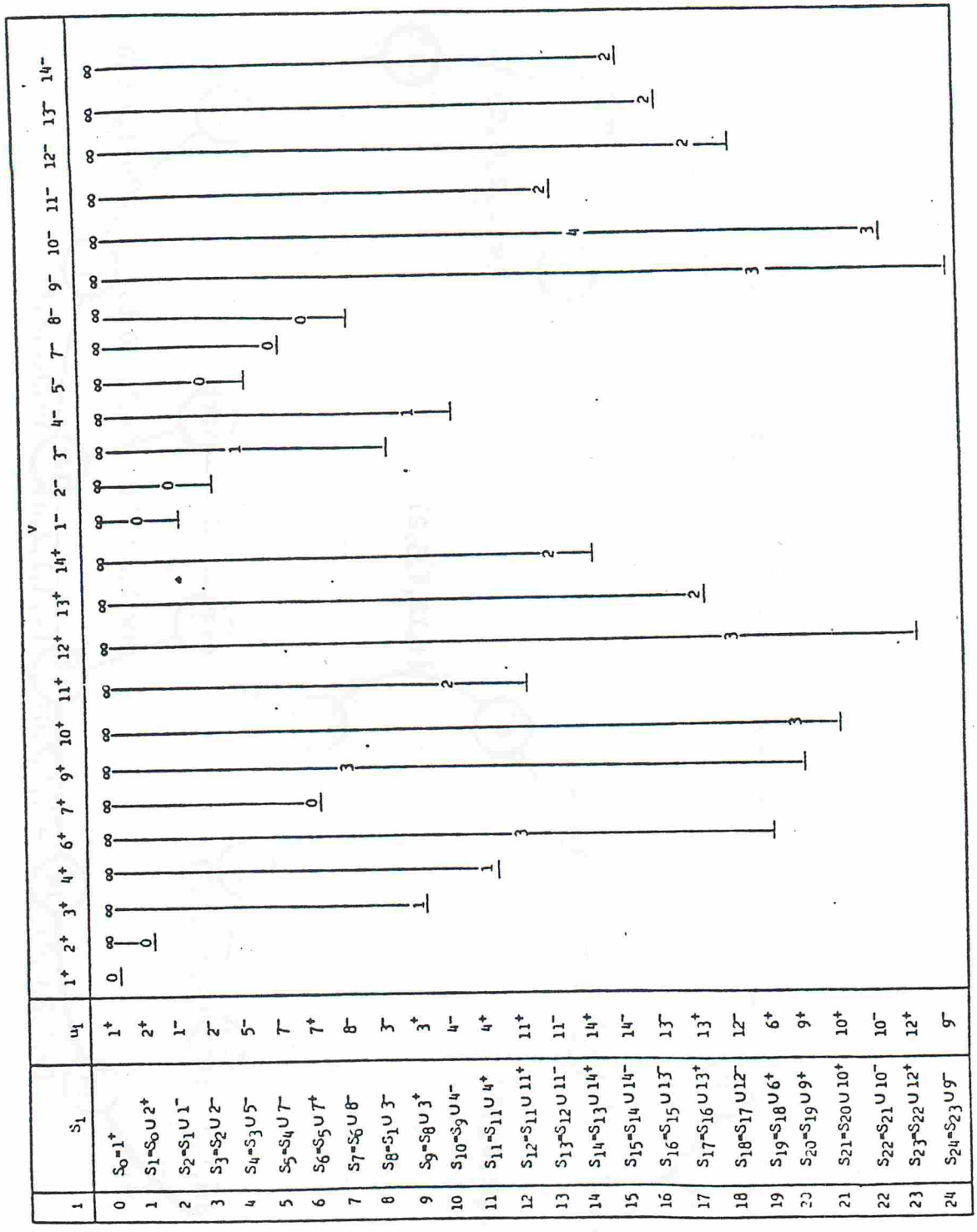


Table 5.4 The execution sequence of Algorithm 5.2 for an example. A node v is selected as the one with the minimum value in the row. Once a node v is selected and put in S_i, it no longer is considered by the algorithm.

Output: Track assignment for each section of a minimum-perturbation path for a net N.

Procedure:

Step 1. Recall that the label of an arc in G is $(x,\delta,\rho,t,\lambda)$ where x, δ, ρ, t, and λ are as defined in Section 5.4. Let the arcs in path P be numbered sequentially in numerical order. Set $i = 0$, $\bar{\lambda} = 0$, $\bar{t} = 0$.

Step 2. Set $i = i+1$. If arc i is a switching arc, go to step 3. Otherwise, set $\bar{\lambda} = \max(\bar{\lambda},\lambda_i)$, and $\bar{t} = \max(\bar{t},t_i)$ and go to step 4.

Step 3. Move the $\bar{\lambda}$ routed paths (with the inner-most path in track $\bar{t}$) in D covering the point p, which is the left point of the switching interval, one track further away from the point line. Route the feasible segment in track $\bar{t}$ (or track 1 if $\bar{t} = 0$). Set $\bar{\lambda} = \bar{t} = 0$. Go to step 2.

Step 4. If all arcs in the path have been examined, exit the procedure since net N has been routed. Otherwise, return to step 2 to examine the next arc in path P. □

5.5.4 An example of the operation of Algorithm 5.3

Using the minimum-perturbation feasible path found in the example of Section 5.5.2 (Fig.5.11), and beginning with arc $(1^+,2^+)$, each arc is examined in turn. The first arc is a section arc with label (+,3,0,0,0). Thus, $\bar{\lambda}$ and $\bar{t}$ remain zero in step 2. The next arc is a switching arc so segment (1,2) in street S^+ is routed in track 1 and no routed sections are moved. The next arc, $(2^-,3^-)$, sets $\bar{\lambda}$ and $\bar{t}$ both to 1 in step 2. The next arc is a switching arc; thus, segment (2,3) is routed in track 1 and street S^-, and the routed segment originally in track 1 is moved to track 2. Continuing this way, the final routing configuration D^* is shown in Figure 5.14.

5.5.5 Timing analysis of Algorithm 5.3

Algorithm 5.3 is a straightforward procedure which operates in linear time since it examines each arc exactly once. If we let d be the number of section arcs and z the number of switching arcs in path P then the number of operations required for Algorithm 5.3 is shown in Table 5.5.

The total number of operations required by Algorithm 5.3 is (summing the number of operations shown in Table 5.5):

$$\mathcal{O}ps = 1+2z+3d-3z+3z+d$$
$$= 2z+4d+1 \ .$$

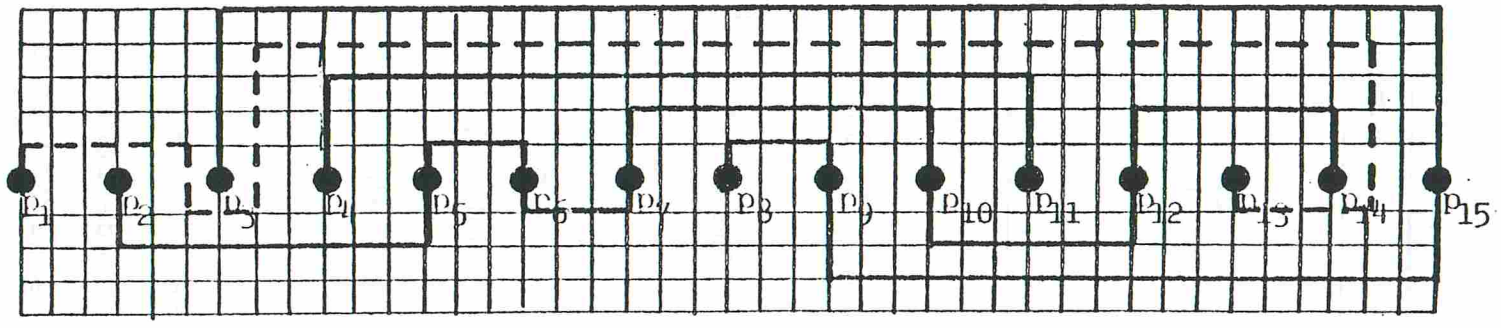


Figure 5.14. The final routed configuration D^* for the example routed configuration D shown in Fig.5.11. N = (1,13) is the net added to D.

Table 5.5. Number of Operations per Step for Algorithm 5.3.

Step	Number of Operations
1	1
2	2z for switching arc 3(d-z) for non-switching arc
3	3z
4	d

For $\mathcal{O}ps_{max}$, $z = \frac{d}{2}$, and $d = 3n-3$, thus

$$\mathcal{O}ps_{max} = 3n-2 + 4(3n-1)$$

$$= 15n - 14 .$$

For $\mathcal{O}ps_{min}$, we assume a minimum number of arcs in a path, namely, one. Then $z = 0$ and $d = 1$ and, hence

$$\mathcal{O}ps_{min} = 5 .$$

5.6 Conclusions

The floating-track routing method describe here is more general than the fixed-track case in that segments assigned to track may be reassigned to different tracks in order to accommodate a new routed path. Whereas the fixed-track routing method lends itself to routing change, the

floating-track method lends itself more to routing a total new configuration.

The timing complexity for the floating-track method is similar to the fixed-track method. The (worst-case) complexity is $\mathcal{O}(n^3 \log n)$ for n-1 nets and n points. However, for realistic configurations (e.g., where the number of tracks in either street is small compared to the number of points in the configuration) the complexity is $\mathcal{O}(n^2)$. Thus, the method given here is practical and requires approximately the same execution time as for the fixed-track case. Chapter 9 gives experimental evidence that this is the case. Also, as for the fixed-track case, if one or more paths exist for a net in a floating-track configuration, the method presented here guarantees that not only will a path be found, but that the chosen path is optimal for any of the goals considered; namely, minimum-length, minimum-congestion, or minimum-perturbation.

CHAPTER 6

MINIMUM SWITCH UNIDIRECTIONAL ROUTING

6.0 Introduction

Special cases of unidirectional routing are considered in this and the next chapter. Specifically, the problem studied in this chapter is that of routing a two-point net N in an unconstrained unidirectional point environment D such that there is a minimum number of switching paths in the feasible path for N in D.

An example of routing problems where the number of switches is to be minimuzed is found in routing problems where each plane has a preferred wire direction. Since each switch is orthogonally directed with respect to horizontal sections of a path, it is necessary to minimize the number of switches in the path.

Although the approaches presented in the previous three chapters of this report can be easily modified to solve the minimum switch problem, the approach described here is more straightforward and efficient. As for the general approaches already discussed, this approach determines a minimum-cost path in a graphical model of the unidirectional configuration. The graph used is called a *simplified access graph*.

6.1 The Simplified Access Graph

Let D be an unconstrained unidirectional point environment with n points. Let x represent "+" or "-". Define a graph $G_s = (V,E)$ where the node set V contains nodes v_i^x. Each node v_i^x represents a maximal subset of intervals $u_{q_1}^x, u_{q_2}^x, \ldots$ along the point line in D such that each such interval in v_i^x is accessible in the street S^x to all others in the subset. Each node is labeled + or - if accessibility is via street S^+ or S^-, respectively, between intervals in the subset represented by the node. The arc set E is defined on V where an arc exists between two nodes u^x and v^x if and only if the interval subset associated with u^x contains at least one interval that is also contained in the interval subset associated with v^x. For the sake of simplicity, the subset of intervals associated with a node v^x is also labeled v^x.

As an example of a simplified access graph, consider the unidirectional point environment D shown in Figure 6.1.

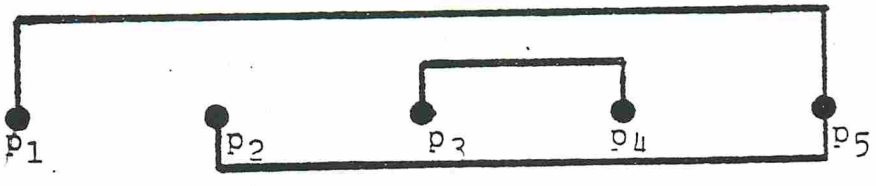


Figure 6.1. An example configuration illustrating interval accessibility.

The nodes defined by D are:

node	intervals
v_1^+	u_1^+, u_2^+, u_4^+
v_2^-	u_2^-, u_3^-, u_4^-
v_3^+	u_3^+
v_4^-	u_1^-

The graph G_s is defined by this environment is shown in Figure 6.2.

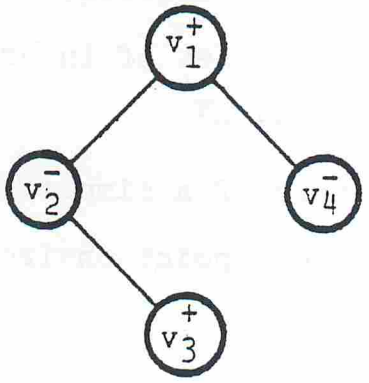


Figure 6.2. The simplified access graph for the routed configuration shown in Fig. 6.1.

We now give an algorithm for the creation of the simplified access graph.

Algorithm 6.1: Simplified access graph construction.

Purpose: This procedure generates a simplified access graph G_s from a routed configuration D containing n points.

Method: The n points in D are scanned twice from point p_1, first with respect to street S^+, and the second with respect to street S^-. At each point p_i, the configuration of p_i and p_{i+1} is determined (i.e., is p_i or p_{i+1} covered, and, if so, is it the first or last point of the segment?). Depending upon the configuration, an arc is created, and a stack operation is performed.

Input: A routed configuration D.

Output: The simplified access graph G_s for D.

Procedure:

Step 1. (initialization) Set x = "+", and clear stack S. Define n-1 locations called NODE. NODE(u) contains the node that contains interval u in street S^x. Set all n-1 NODE locations to 0. Set a single location called VERTEX to 0.

Step 2. Set i = 0. Push item (0) on stack S. (Even though there is no point p_0 in D, we need this dummy item in S to ensure correct processing of the end points in D by the algorithm.) Set

VERTEX = 0.

Step 3. (main loop) Let the top item in stack S be (j). Set i = i+1. Consider points p_i and p_{i+1} in D. Select the applicable case from Table 6.1 and branch to the particular step as indicated.

Step 4. If i = 1, then go to step 7. Otherwise, go to step 5.

Step 5. If x = "+", then set NODE(i) = j. Otherwise, create an arc between the node labeled "+" and numbered NODE(i), and the node labeled "-" and numbered j. Go to step 9.

Step 6. If x = "+", then set NODE(i) = j. Otherwise, create an arc between the node labeled "+" and numbered NODE(i), and the node labeled "-" and numbered j. Pop S. Go to step 9.

Step 7. Set VERTEX = VERTEX+1. Create a node numbered VERTEX and labeled x. If x = "+", then set NODE(i) = VERTEX. Otherwise, create an arc between the node labeled + and numbered NODE(i), and the node labeled "-" and numbered VERTEX. Push VERTEX on stack S. Go to step 9.

Step 8. Set VERTEX = VERTEX+1. Create a node labeled x and numbered VERTEX. Go to step 5.

Step 9. If i ≠ n-1 then go to step 3. Otherwise, go to step 10.

Step 10. If x = "+", then set x = "-" and go to step 2. Otherwise, exit algorithm. □

Table 6.1. Jump Table for Algorithm 6.1

IF p_i is Covered in S^x	IF p_{i+1} is Covered in S^x	IF p_i is Left Point of Segment	IF p_{i+1} is Left Point of Segment	THEN Go To Step:
no	no	—	—	4
yes	no	no	—	5
no	yes	—	yes	4
no	yes	—	no	6
yes	yes	no	no	6
yes	yes	no	yes	5
yes	no	yes	—	7
yes	yes	yes	yes	7
yes	yes	yes	no	8

We next show the characteristics of graph G_s.

First we show that graph G_s is a bipartite unidirected graph.

Theorem 6.1: Given any routed configuration D, the simplified access graph G_s for D is a bipartite graph.

Proof: By definition, any interval in D occurs in exactly one + node and one - node. Also by definition, all intervals accessible to each other in a particular

street occur in the same node. Thus, no arc connects any like-labeled nodes. Since there are only two sets of like-labeled nodes in G_s, G_s is bipartite. □

The size of G_s is of interest since the processing time of any algorithm which processes G_s is directly related to the size of G. The following theorems relate the size of G_s to the number n of points in D and the number c of routed segments in D, thereby showing that G_s is no larger than a linear function of n.

Theorem 6.2: Given a routing configuration D with c routed segments on a unidirectional point set $P = \{p_1, p_2, \ldots, p_n\}$ with the points p_1 and p_n at the extreme ends of the line of points, and a simplified access graph G_s with m nodes, then $m = c+1$ for $c \geq 1$, if a segment between p_1 and p_n exists, or $m = c+2$ for $c \geq 0$ otherwise.

Proof: We will prove this theorem by induction.

Case 1 Assume a segment between p_1 and p_n exists and no other segment exists. Then there are two nodes in G_s and one segment in D. Thus, for $c = 1$, $m = c+1$ holds. Let $c = 2$ and let one of the two segments have end points p_1 and p_n. Then an additional node exists in G_s. Finally, assume $m = c+1$ for m nodes

and c segments, and let a segment be added to D in S^+. Then one additional node is created in G_s (i.e., since segments cannot cross each other in S^+, and two segments cannot have the same end point, then the addition of a segment creates an additional node in G_s). The same argument holds for S^- as well; i.e., adding a segment in S^- creates an additional node in G_s.

Thus, by induction, the number of nodes in G_s is one more than the number of segments in D for $m \geq 1$, or $m = c+1$.

Case 2 Assume no segments exist in D. Then G_s contains two nodes. Thus, if $c = 0$ then $m = 2$, and $m = c+2$ is satisfied. Now assume $m = c+2$ and add a segment in S^+ (or S^-) such that the added segment does not have p_1 and p_n as end points. Then one additional node is added to G_s. Therefore, case 2 is proven by induction and the number of nodes in G_s is two more than the number of segments in D assuming no segment exists with end points p_1 and p_n. □

Lemma 6.1: Given an unconstrained routing configuration D with n unidirectional points, the number of nodes m in G_s is less than or equal to n+1.

Proof: The maximum number of segments that can be routed in a given street on n points in n/2. If the

maximum number of segments is routed in a street, then no more than one less than the maximum number n/2 of segments can be routed in the opposite street. If, in fact, n/2 segments were routed in both streets, a closed loop occurs and thus one of the routed segments is unnecessary (see Figure 6.3). Therefore, the maximum number of segments c routed in a configuration is n-1. From Theorem 6.2, $m \leq c+2$. Thus, $m \leq n+1$. □

Figure 6.3. Example of a closed loop of routed segments. Removal of any one segment still leaves all points electrically connected.

Theorem 6.3: Consider a routed configuration D with n points and c segments. Let k be the number of arcs in the access graph G_s for D. Then $c \leq k \leq n-1$ for $c \geq 1$. $k = 1$ if $c = 0$.

Proof: ($c = 0$) If there are no segments in D, then there are two nodes in G_s with one arc between them by definition of G_s. Thus $k = 1$.

(lower bound, $c > 0$) Consider c routed segments in D and k arcs in G_s. Adding one more routed segment to D adds at least one more arc in G_s. Now assume each one of the c+1 segments added k+1 arcs, or $c = k$. Since this is the minimum, $k \geq c$.

(upper bound, $c > 0$) There are n-1 intervals in a unidirectional configuration with n points. Since each arc in G_s could be uniquely constructed by one and only one interval in D, then there can be no more than n-1 arcs in G_s. □

6.2 Minimum-Switch Paths

In this section we consider adding a net $N = (s,t)$ between two points s and t in a configuration D. Stree capacities are ignored. We assume the path for N may cross from one street to the other via switches. It is desired to minimize the number of such switches in order to keep the interval congestion down.

A characteristic of the algorithm presented here is that the path is not restricted to be of minimal length. In fact, a path may "double back" on itself if required in order to complete the route successfully. We define a node

in G_s to be an s-node if it contains interval u_s. Likewise, a t-node in G_s contains interval u_t. Note that G_s may contain two s-nodes and two t-nodes depending on whether or not p_s or p_t are uncovered. To route net N = (s,t), at least one s-node and one t-node must exist.

To motivate the minimum-switch path algorithm presented in the next section, consider the configuration D shown in Figure 6.4.

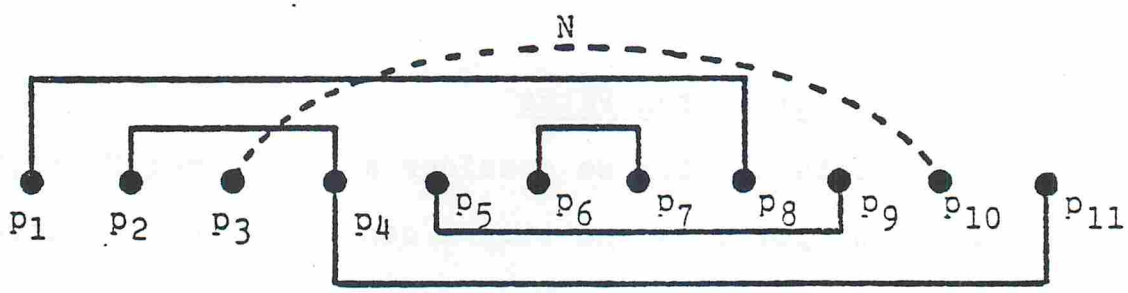


Figure 6.4. Example routing configuration illustrating addition of a minimum-switch path in an unconstrained configuration.

It is desired to add net N = (3,10) to the configuration. First, we generate from D the access graph G_s as shown in

Figure 6.5. Then, by finding the shortest path between either one of the s-nodes (s is point p_3) and one of the t-nodes (t is point p_{10}), we have in G_s a representation of the path in D which contains the minimum number of switches for N. Algorithm 6.2 implements this procedure. The shortest path in G_s, unique in this case, is v_5^-, v_1^+, v_6^-. Since there are two arcs in this path, there are two switching paths in the routed path for N in D. There are no paths with fewer than two switching paths for N. Figure 6.6 illustrates the routed path in D.

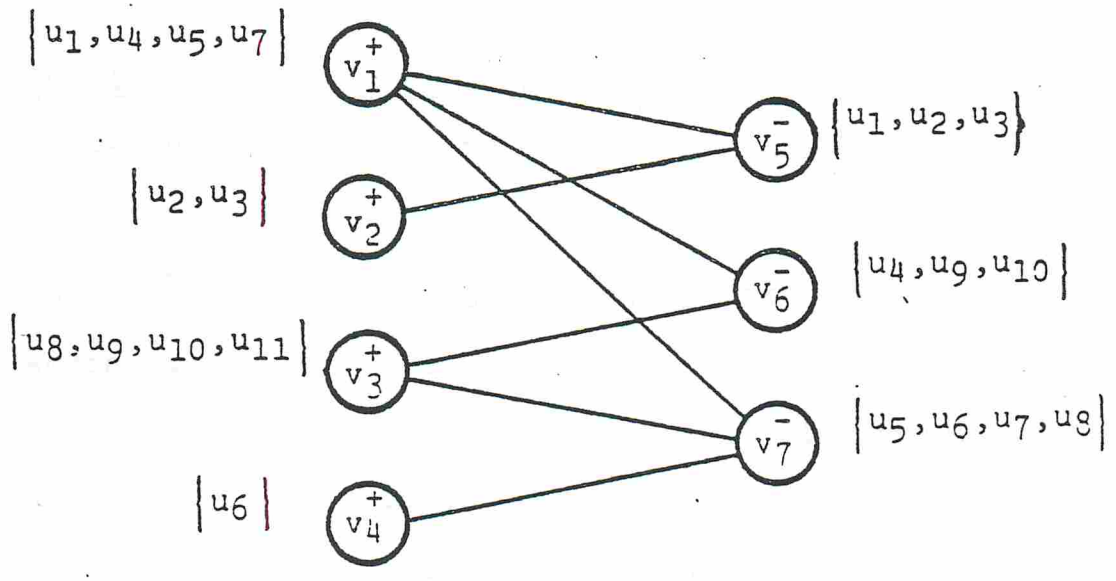


Figure 6.5. Access graph G_s for example in Figure 6.4.

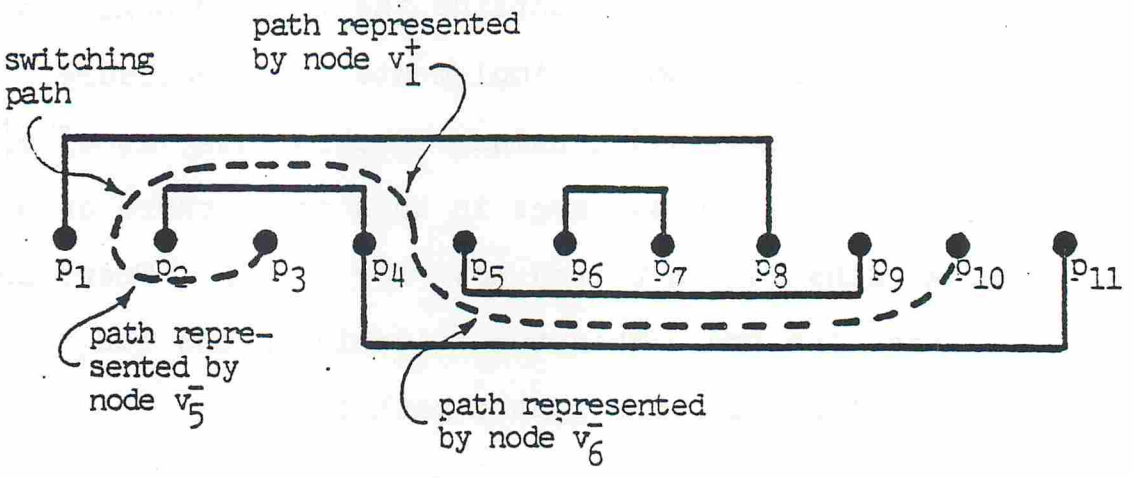


Figure 6.6. Configuration showing feasible path as produced by Algorithm 6.2.

6.2.1 Algorithm 6.2. Minimum-switch path algorithm

The following algorithm finds a minimum-switch path in G_s.

Purpose: This algorithm determins a path for net N = (s,t), point p_s to the left of p_t, in an existing unconstrained routed unidirectional point environment D such that the number of switching paths is minimized.

Method: This algorithm determines a path in a simple access graph G_s for a particular routing configuration D. The path found directly identifies the entire minimum

switching path in D for N.

Procedure:

Step 1. Generate the access graph G_s for D using Algorithm 6.1.

Step 2. If u_s and u_t are both in the same node v in G_s, then the path between p_s and p_t does not switch between streets. The street containing the segment for N is in S^+ if v is labeled "+" or S^- is v is labeled "-".

Step 3. If u_s and u_t are not both in the same node, then obtain the shortest path P between p_s and p_t using Dijkstra's Algorithm (see Section 2.11, Chapter 2). This path represents a feasible path for N in D such that a minimal number of switches are used. If no path is found in G_s, then there exists no feasible path for N in D. Each node in P represents a segment in D, and the arcs represent switches. The label of the node identifies the street in which each subpath lies. To determine the location of the switching paths and, thus the end points of each segment, let path P in G_s consist of z nodes $v_{q_1}, v_{q_2}, \ldots, v_{q_z}$ with z-1 arcs $(v_{q_1}, v_{q_2}), (v_{q_2}, v_{q_3}), \ldots, (v_{q_{z-1}}, v_{q_z})$. Each node v_{q_i} consists of a set

of one or more intervals a_j. Then a feasible switching path is represented by any one of the intervals in $v_{q_i} \cap v_{q_{i+1}}$. □

For the example in Figure 6.3, $P = (v_5^-, v_1^+, v_6^-)$. To determine the location of each switch take the pairwise intersection of the sets of intervals represented by each node and its successor in the path P. Thus,

$$v_5^- \cap v_1^+ = \{p_1\}$$

$$v_1^+ \cap v_6^- = \{p_4\}.$$

Referring to Figure 6.5, intervals u_1 and u_4 do, in fact, contain the switching paths for the routed path N.

6.2.2 Timing analysis of Algorithm 6.2

The timing analysis of Algorithm 6.2 is rather straightforward to obtain. Generating the access graph in step 1 requires $\frac{15}{2}n+4$ operations. Testing for the equivalence of the s-node and t-node in step 2 is a unit operation.

Step 3 uses Dijkstra's Algorithm presented in Chapter 2 which requires a maximum of

$$\mathcal{O}ps_d \leq 3m+(m-1)\log(m-1) + \sum_{i=0}^{m} d(u_i)[1+\log(m-i-1)] \text{ operations}$$

where m is the number of nodes in the simplified access graph. This expression also assumes that the path found by the algorithm passes through all nodes in the graph.

The maximum number of nodes m in the graph is $m = n+1$ where n is the number of points in the configuration from which the graph is obtained. The only configuration which gives this value is shown in Figure 6.7.

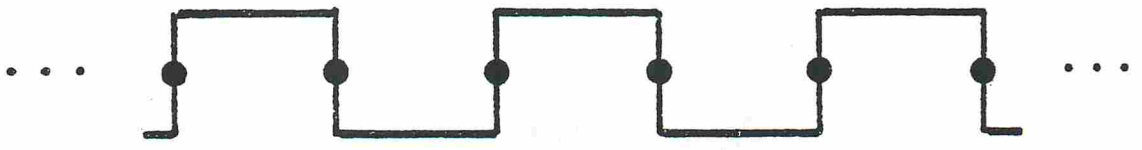


Figure 6.7. The configuration which produces the maximum number of nodes in the simplified access graph.

The access graph contains n+1 nodes each of which have degree $d(u_i) = 1$ except for two nodes which each have degree $\frac{n-1}{2}$ if n is odd, or one has degree $\frac{n}{2}$ and the other has degree $\frac{n}{2}-1$ if n is even.

Substituting $m = n+1$, and noting that the two nodes with degree greater than one contribute the greatest number of

operations to the above expression when i = 0 and 1 in the summation, then

$$\mathcal{O}ps_d \leq 3n+3+n\log n+\frac{n-1}{2}(1+\log n) + \frac{n-1}{2}(1+\log(n-1)) + \sum_{i=2}^{n+1}(1+\log(n-i))$$

$$< 5n+1+(2n-1)\log n + (\log n!-\log(n-1)) .$$

Using Sterling's approximation for n!,

$$\log n! < \frac{4n+1}{2}\log n - \frac{1}{2}n .$$

Then,

$$\mathcal{O}ps_d < \frac{1}{2}[9n+2+(8n-1)\log n-\log(n-1)] .$$

Adding the number of operations for steps 1 and 2, we obtain the total maximum number of operations, $\mathcal{O}ps_{max}$, required by Algorithm 6.2. Thus

$$\mathcal{O}ps_{max} = \frac{1}{2}[24n+12+(8n-1)\log n-\log(n-1)] ,$$

and Algorithm 6.2 is of $\mathcal{O}(n\log n)$ complexity.

CHAPTER 7

ROUTING WITH NO SWITCHES IN A UNIDIRECTIONAL CONFIGURATION

7.0 Introduction

In this chapter we consider the second of the three special cases of unidirectional routing. The case considered here is to route nets in a unidirectional point environment where switches are not permitted. Thus, a routed two-point net is equivalent to a routed segment. The point line consists of points only from the original point set P. The objective function for which the results described here apply is to minimize the number of existing routed segments which must be moved to the opposite street to accommodate a new feasible segment. This objective function differs from that considered in Chapter 5 where routed sections are moved to adjacent tracks in the same street whereas in this chapter segments are moved to the opposite street. Actual track assignment is not considered. Hence, there are no track constraints.

After a brief discussion on the types of connections at a point that are allowed in this chapter, the concept of a conflict graph is introduced. Using the fact that a con-

flict graph for a routed configuration is bichromatic, the method of determining the path for a new net is straightforward and efficient.

7.1 Tree Connections

Up to this point, all connections considered at points in a configuration have been *chain* connections, i.e., two segments meeting at a common point must be routed in opposite streets. A *tree* connection is one where two segments meeting at a common point are routed in the same street. We further restrict the two adjoining sections to meet at the same point as illustrated in Figure 7.1.

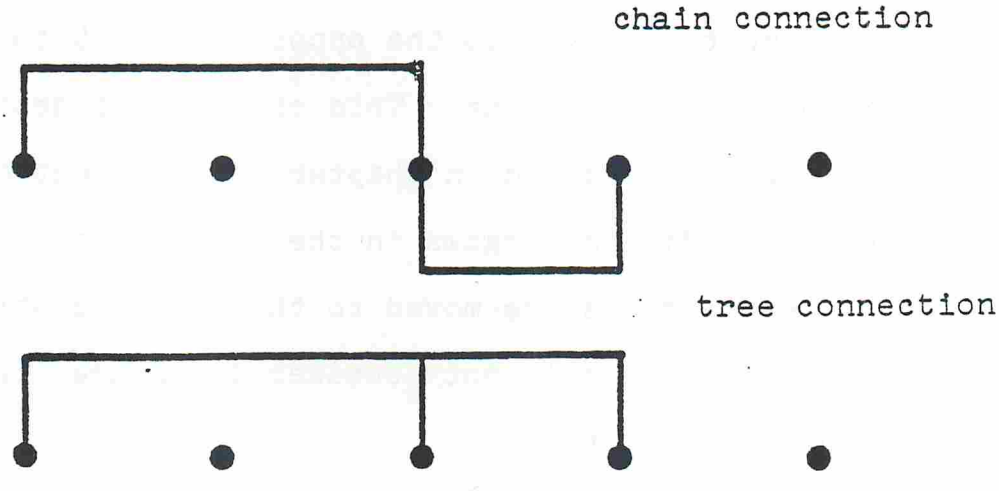


Figure 7.1. Routed configurations illustrating a chain connection (top) and a tree connection (lower).

There is a benefit to considering tree connections since they enhance the probability that a new net will be successfully routed. As an example, consider the routed configuration shown in Figure 7.2. Assume net $N = (p_3, p_5)$ is to be routed. If only chain connections are permitted, N cannot be routed. However, if tree connections are permitted, segment (p_4, p_6) can be moved to form a tree connection at point p_4 and the new net can be routed in the lower street as shown in Figure 7.3.

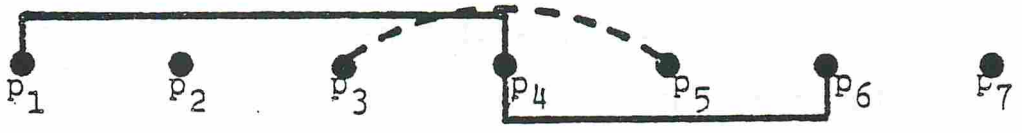


Figure 7.2. Since tree connections are not permitted, net (p_3, p_5) cannot be routed.

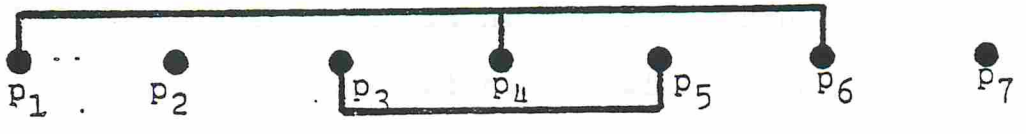


Figure 7.3. With tree connections permitted, net (p_3, p_5) can be routed.

7.2 The Conflict Graph

Let a *conflict graph* H for a routed configuration D be an undirected graph (V,E) where V is a set of nodes, each representing a two-point net in D, and E is a set of arcs where an arc a in E joins nodes u and v if and only if

1. one of the two end points of the two-point net represented by u lies between the end points of the two-point net represented by v; or
2. the two nets corresponding to u and v share a common point which has been prespecified to be a chain connection.

Note that if two two-point nets N_1 and N_2 have a common point such that a tree connection is permitted, then no arc is inserted between nodes N_1 and N_2 in H.

Finally, each node is labeled "+" if the two-point net represented by the node is in the street S^+, or "-" if the two-point net represented by the node is in the street S^-. This labeling imposes a bicoloring on H thereby implying that H is bichromatic.

As an example of a conflict graph, consider the routed configuration shown in Figure 7.4a. The conflict graph for this configuration is shown in Figure 7.4b. Assume only chain connections are permitted.

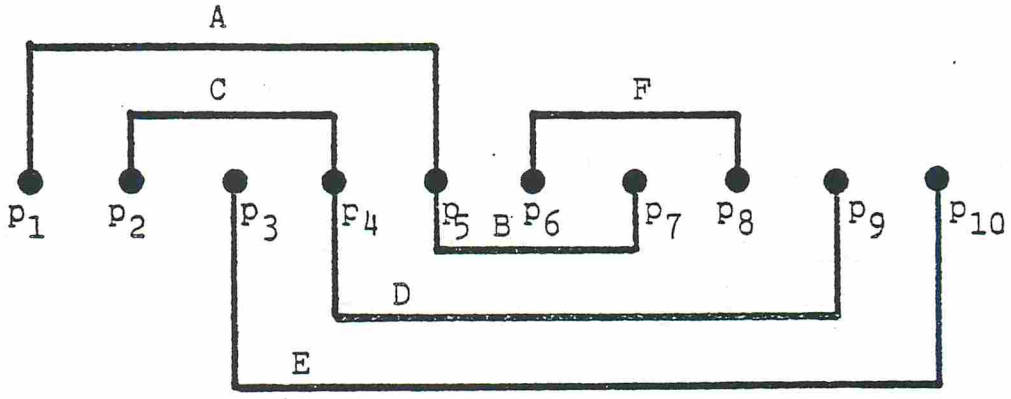
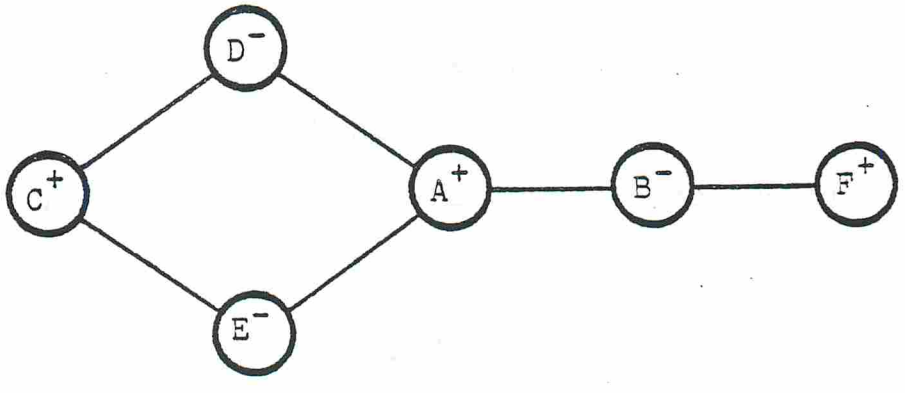


Figure 7.4a. A routed configuration. No switching paths are allowed.



7.4b. Conflict graph for the routed configuration shown in Fig.7.4a. (An arc in the graph implies that nets represented by the two adjacent nodes must be routed in opposite streets.)

7.2.1 Generating the conflict graph

The following graph generation algorithm assumes that either 1) only chain connections are allowed, or 2) both tree and chain connections are permitted. Also, all nets to be routed are two-point nets.

Algorithm 7.1: Creating a conflict graph

Purpose: This procedure creates a conflict graph H for a routed configuration D. D contains n points. No switching paths are permitted.

Method: Two stacks, U^+ and U^- are used to store two-point nets in street S^+ and S^- respectively as they are encountered in a left-to-right scan of the point line. A pointer associated with each net points to the top of the opposite stack at the time the two-point net is pushed on its appropriate stack. When the two-point net is popped, an arc is created between the node in H corresponding to that net and each node in H corresponding to two-point nets in the opposite stack down to the depth pointed at by the pointer associated with the popped net.

Input: A routed configuration D.

Output: A conflict graph H for D.

Define the operation PUSH (N,x) to be:

push net N and PTR, where PTR is the depth of stack $U^{\bar{x}}$, onto stack U^x, and create node V_N^x in H.

Define the operation POP (N,x) to be:

pop stack U^x to get N and PTR. Let the nets in stack $U^{\bar{x}}$ from the top of $U^{\bar{x}}$ down to PTR be a,b,c,... . Create arcs $(V_N^x, V_a^{\bar{x}})$, $(V_N^x, V_b^{\bar{x}})$, $(V_N^x, V_c^{\bar{x}})$,... in H.

Step 1. Let p_i be the left-most covered point in the line of points. Go to step 3.

Step 2. Let p_i be the next covered point in the point line scanning from left to right. If there are no more covered points, exit procedure. If p_i is covered by only one routed net, go to step 3. Otherwise go to step 4.

Step 3. Let the net η covering p_i be in street S^x. If p_i is the left point of the net, PUSH (η,x). If p_i is the right point of the net, POP (η,z). Go to step 2.

Step 4. (a) Let the two nets covering p_i be η_1 and η_2. If both nets are in the same street S^x let η_1 be the net with p_i as its right point. POP (η_1,x) and then PUSH (η_2,x) and go to step 2. If both nets are not in the same street then do the following: (for all cases let η_1 be in street S^x).

(b) If p_i is the right point of one of the nets

and the left point of the other, let the nets be η_1 and η_2 respectively. Then, if tree connections are note permitted, PUSH $(\eta_2,\bar{x})$ and then POP (η_1,x). If tree connections are permitted, POP (η_1,x) and then PUSH $(\eta_2,\bar{x})$. Go to step 2.

(c) If p_i is right point of both nets, let PTR_1 be the pointer on top of stack U^x. If tree connections are not permitted and PTR_1 is less than the depth of $U^{\bar{x}}$, or tree connections are permitted and PTR_1 is greater than the depth of $U_1^{\bar{x}}$, POP (η_1,x) and then POP $(\eta_2,\bar{x})$. Otherwise, POP $(\eta_2,\bar{x})$ and then POP (η_1,x). Go to step 2.

(d) If p_i is the left point of both nets, PUSH (η_1,x) and then PUSH $(\eta_2,\bar{x})$. Set PTR in the top of $U^{\bar{x}}$ to the depth of U^x minus one rather than to just the depth of U^x. Go to step 2.

7.2.2 Timing analysis of Algorithm 7.1

If a stack PUSH or POP, and arc or node creation are each assumed to be unit operations, then the operation PUSH (N,x) requires two operations, and POP (N,x) requires 1+d operations where d is the depth to PTR in stack $U^{\bar{x}}$.

It is apparent that maximizing d maximizes the number of operations required by Algorithm 7.1. In other words, maximizing the number of routed nets that conflict in any configuration maximizes the run time of the algorithm. At most, n/4 nets can conflict with another as illustrated by the configuration shown in Figure 7.5.

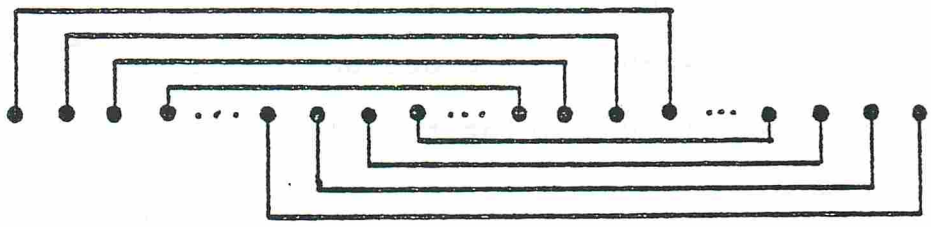


Figure 7.5. An example of a configuration which maximizes the number of operations required by Algorithm 7.1 to generate the conflict graph H.

There are at most $\frac{1}{2}$n routed nets in such a configuration. The maximum number of conflicting nets occurs when half of the nets are in the upper street and half are in the lower

street. Thus, as the algorithm processes, $\frac{1}{2}n$ PUSHs are executed in step 3, one each for the first n/4 points attached to nets routed in the upper street. The second n/4 points are attached to the lower nets. A PUSH is executed in step 3 for each of these points. The third n/4 points are attached to the upper nets which conflict with all of the lower nets. Thus, step 3 is executed n/4 times with a POP occurring each time. Finally, the last n/4 points are attached to the lower nets. A POP is executed in step 3 for each of these last n/4 points.

Half way through the processing of the configuration in Figure 7.5, each stack U^+ and U^- has a depth of n/4. As the third n/4 points are processed, the depth decreases by one for each point processed in stack U^+. The same thing happens in U^- during the processing of the last n/4 points. Thus,

$$\begin{aligned} Ops_{max} &= 2(\tfrac{n}{4}) + 2(\tfrac{n}{4}) + (1+\tfrac{n}{4}) + (1+\tfrac{n}{4}-1) + (1+\tfrac{n}{4}-2) \\ &\quad + \cdots + (2) + (1+\tfrac{n}{4}) + (1+\tfrac{n}{4}-1) + (1+\tfrac{n}{4}-2) \\ &\quad + \cdots + (2) \\ &= n + 2\sum_{i=1}^{n/4} (2+\tfrac{n}{4}-i) \\ &= \frac{n^2}{16} + \frac{7n}{4}. \end{aligned}$$

7.2.3 The size of H

Let b be the number of two-point nets routed on D, n be the number of unidirection points in D, and k the number of arcs in graph H. By definition, there are b nodes in H.

The maximum number of two-point nets that can be routed in D is n/2. It will be shown later in Section 7.3.1 that the graph H is bipartite. Thus, the maximum number of arcs in H occurs if H is a complete bipartite graph with the same number of nodes in each non-adjacency node set of H. Thus, if H has n/2 nodes, then H has $k = (\frac{n}{4})(\frac{n}{4}) = \frac{n^2}{16}$ arcs in the worst case.

7.3 Finding a Non-Switching Path

In the discussion that follows, we consider a *feasible solution* to a unidirectional point routing problem to be defined as before; namely, a path in the routed environment has been found, but the path has not been assigned to a track. A *routed solution* has been obtained from a feasible solution when the feasible path is assigned to a track. Only feasible solutions are considered in this chapter; routed solutions require a track assignment procedure which is left for further study.

7.3.1 Theory

The following theorems are presented since they imply a procedure which minimuzes the number of routed nets which must be assigned to the opposite street to accommodate a

feasible path for a new net. All of the theorems assume a conflict graph constructed per Section 7.2.1 of this chapter.

Theorem 7.1: Let H be a conflict graph representing a non-switching two-point net unidirectional routing problem P. Then H represents a feasible solution to P if and only if H is bichromatic.

Proof: (if) Let H be bichromatic. Then each node in H is labeled such that no two adjacent nodes have the same label. From the definition of H, two non-adjacent nodes represent nets that do not intersect if routed in the same street. Thus, if all nets represented by nodes in H having the same label are routed in the same street, and all remaining nodes represent nets routed in the opposite street, no intersections will occur, and thus, a feasible solution results.

(only if) Let H be a conflict graph for a feasible solution to a unidirectional routing problem. Then all nets routed in the same street are non-intersecting and are represented by non-adjacent nodes in H. These nodes have the same label. For the nets in the other street, the corresponding nodes have the opposite label. These nodes are also not

adjacent to each other. Since adjacent nodes in H represent only nets routed on opposite sides of the point line, H is bichromatic. □

Corollary 7.1: Let H be a conflict graph for a non-switching two-point net unidirectional routing problem. Then H represents a feasible solution if H has no odd cycles.

Proof: By a well known theorem in graph theory, a graph H is bichromatic if H has no odd cycles [27]. □

The next two theorems require the definition of a graph H′ derived from the conflict graph H as follows. Let H′ be the graph H plus an additional node n representing the two-point net N to be routed in D. Add arcs a_i, i = 1,...,q to H′ between nodes v_i and n, i = 1,...,q if one end of N is within the span of the two-point net in D represented by node v_i.

Thoerem 7.2: Let H be a conflict graph representing a routed configuration D and let H′ be a graph as described above, where node n represents a two-point net N to be routed in D such that no switching paths are permitted. H′ represents a feasible solution to the problem of routing N in D if there are no odd-length paths between any nodes $v_1, v_2, \ldots, v_q$ in H.

Proof: Let the nodes $v_1, v_2, \ldots, v_q$ in H that are adjacent to n in H′ be caleld set V. Assume there is a path of m arcs in H between two nodes u and v in V. (This path exists in both H and H′.) Since nodes u and v are also adjacent to n in H′, then there exists a cycle c of length m+b where b is 1 or 2 depending on whether nodes u and v are adjacent or not. If u and v are adjacent (i.e., b = 1), then there is an odd-length cycle consisting of u, v, and n, and, by Corollary 7.1, H′ is not bichromatic. Thus, b must equal two and cycle c is of length m+2. If there are no odd-length paths between any nodes in V and H then m is always even, and the addition of nodes n to H to create H′ induces a feasible solution. However, if there is some path in H between two nodes in V that is of odd length, then m+2 is odd also, and a cycle of odd length is in H′. Thus, by Corollary 7.1, H′ is not bichromatic, and H′ does not represent a feasible solution. □

Theorem 7.3: Let H be a conflict graph representing a routed configuration D and let H′ be the graph derived from H as described above where node n represents a two-point net N to be routed in D. Let the set V consist

of the nodes $v_1, v_2, \ldots$ adjacent to node n in H'. Then any two nodes v_i and v_j in set V are in different graph components of H if the label of v_i is opposite to the label of v_j for $i \neq j$.

Proof: By Theorem 7.2, there are no odd-length paths between any two nodes in V in H. Assume node v_1 is labeled "+" and v_j is labeled "-", where v_i and v_j are in V. Since all nodes lying in a path between v_i and v_j are bicolored, the path must be of odd length. Thus, there is no path connecting v_i and v_j in H, and therefore, v_i and v_j must be in separate components of H. □

7.3.2 Algorithm 7.2 Non-switching unidirectional path algorithm

The central algorithm of this section is now presented. The theorems and corollaries presented above imply and directly support this algorithm.

Purpose: The algorithm routes a net N in a routed configuration D. No paths are permitted in the switching channels of D. This algorithm assumes that N is a two-point net, and it obtains a path for N in D such that a minimum number of existing routes are assigned to opposite streets.

Procedure:

Step 1. If the conflict graph H does not already exist, create it using Algorithm 7.1. Create conflict graph H′ as defined in Section 7.3.1.

Step 2. Find all components of H (see Algorithm 5.3 in[28]). If any two oppositely labeled nodes in V are in the same component, then exit since a non-switching path does not exist for net N in D.

Step 3. Separate the components of H into two sets H^+ and H^- such that H^+ contains those components that have nodes in V labeled "+", and H^- contains those components that have nodes in V labeled "-". Let π^+ be the number of nodes in the components in H^+ and π^- be the number of nodes in the components in H^-.

Step 4. Update graph H′ to a new conflict graph H reflecting the routed configuration containing the path for the net N as follows: label node n in H′ as "+" if $\pi^+ < \pi^-$, as "-" if $\pi^+ > \pi^-$, and either "+" or "-" if $\pi^+ = \pi^-$. Reverse all labels on all nodes in the components of the set with the smaller number

of nodes (this implies all routed nets represented by the relabeled nodes are reassigned to the opposite street they currently occupy. The label of node n identifies the street in which net N will be routed. □

7.3.3 Examples of the use of Algorithm 7.2

Consider the routed configuration D_1 shown in Figure 7.6a. The conflict graph H_1 for D_1 is shown in Figure 7.6b. Assume that a net $N = (p_1, p_6)$ is to be routed. The graph H_1' which includes node n is shown in Figure 7.6c. The set V is $\{C^-, D^-\}$. Since the nodes in V are all labeled "-", then node n is labeled "+". Thus, net N has a feasible path in street S^+, and this example requires no reassignment of routed nets to accommodate N.

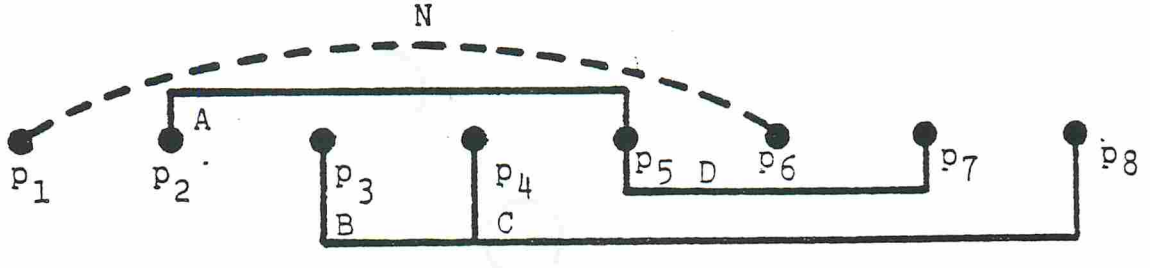


Figure 7.6a. Routed configuration D_1 used to illustrate non-switching routing.

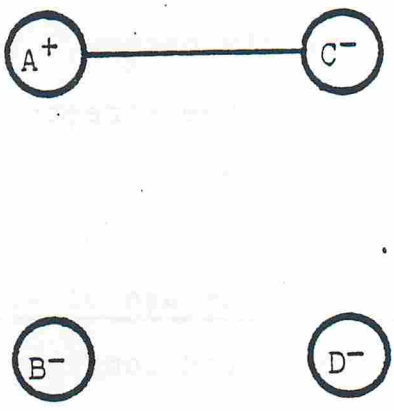


Figure 7.6b. Conflict graph H_1 for routed configuration D_1.

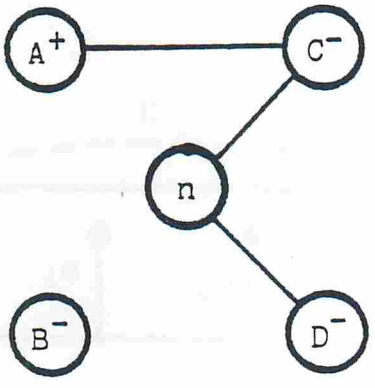


Figure 7.6c. Conflict graph H_1'.

As an example of a routing problem where routed nets must be moved to accommodate a new net N, consider the routed configuration D_2 shown in Figure 7.7a. The conflict graph H_2 for D_2 is shown in Figure 7.7b. The graph H_2' which includes new node $N = (p_4, p_8)$ is shown in Figure 7.7c. The set V is $\{B^+, C^+, E^-\}$. Since the nodes are not all labeled the same, graph H_2 is partitioned into components as shown in Figure 7.7d.

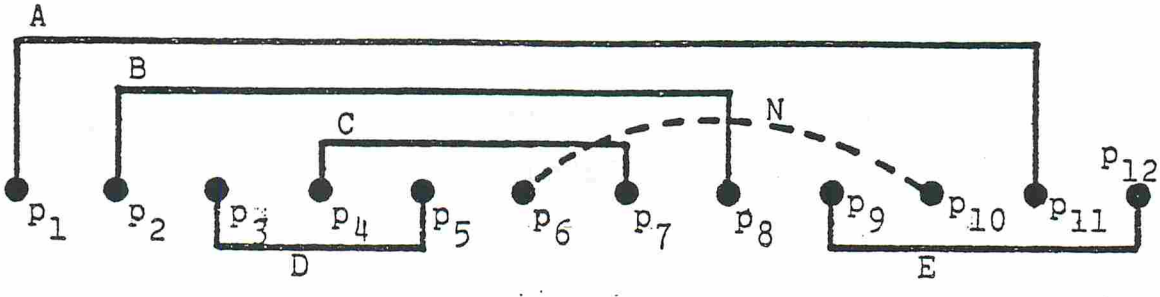


Figure 7.7a. Routed configuration D_2 used to illustrate non-switching routing.

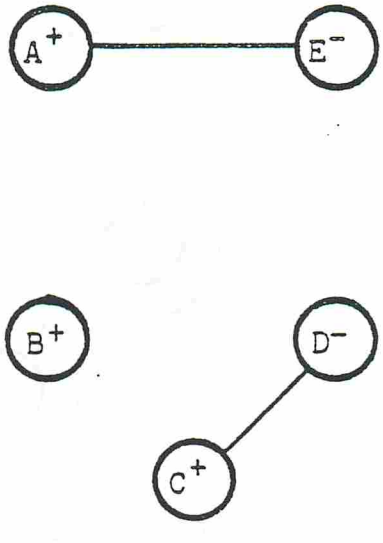


Figure 7.7b. Conflict graph H_2 for routed configuration D_2.

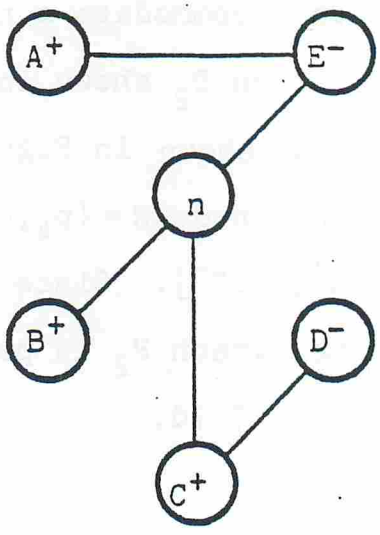


Figure 7.7c. Conflict graph H_2' with node N unlabeled.

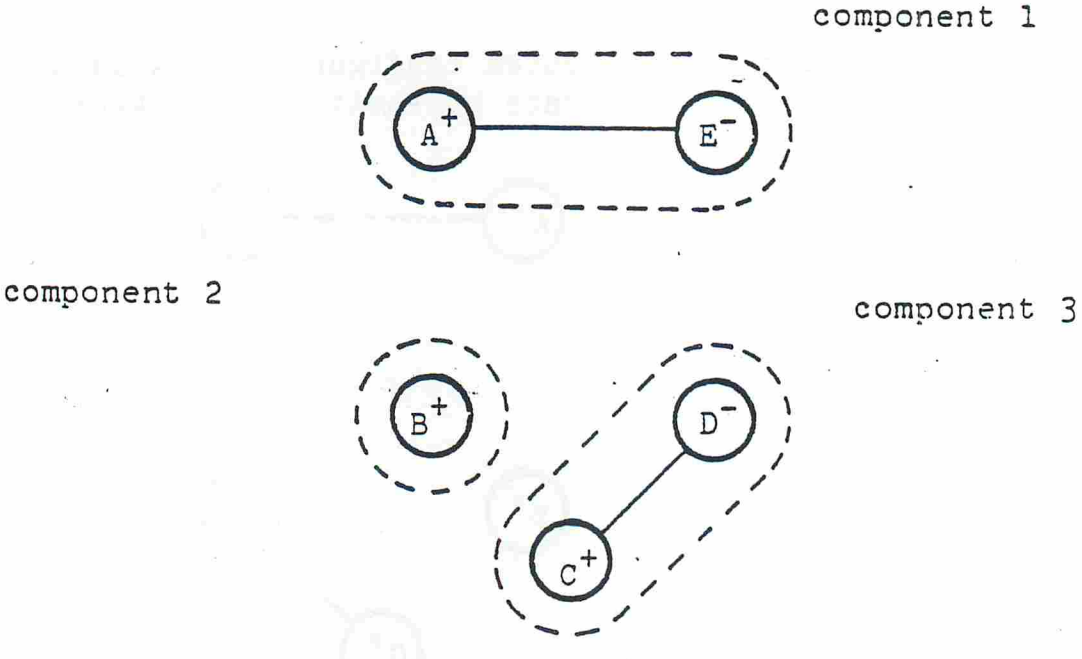


Figure 7.7d. Conflict graph H_2 partitioned into components.

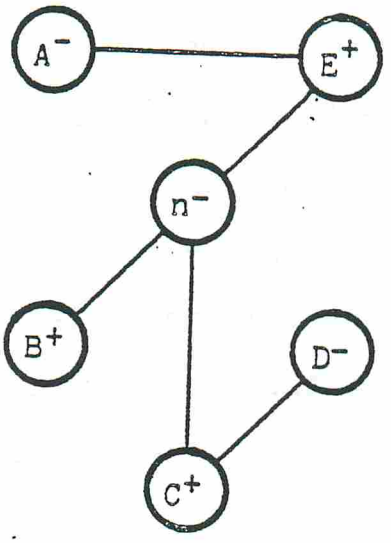


Figure 7.7e. The new conflict graph with node n labeled.

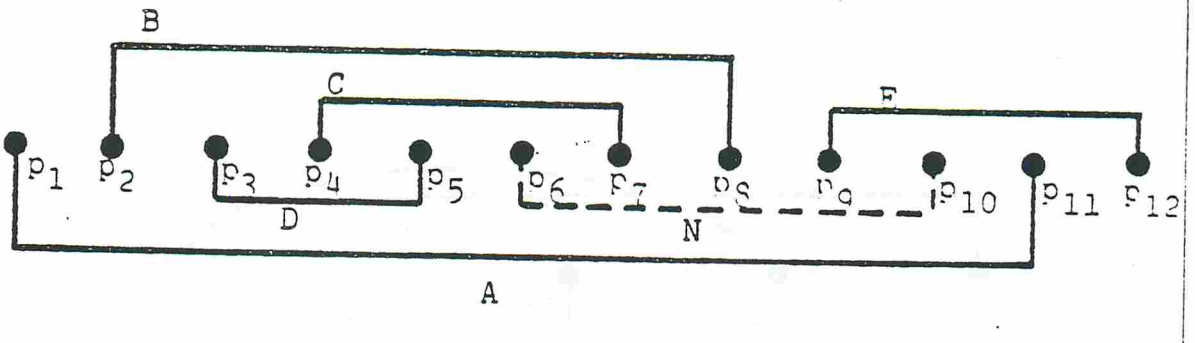


Figure 7.7f. The new configuration containing a path for net N. Two paths (A and E) were moved to accommodate net N.

Nodes in $V = \{B^+, C^+, D^+\}$ are labeled both + and -. Thus, the components of H_2 are separated into two groups H_2^+ (containing components 2 and 3) and H_2^- (containing component 1). The number of nodes π^+ in H_2^+ is 3 and the number of nodes π^- in H_2^- is 2. Since $\pi^- < \pi^+$ node n is labeled "-" and each node in component 1 is relabeled. The new conflict graph which represents the configuration D_2 with a path for N is shown in Figure 7.7e. The new configuration is shown in Figure 7.7f.

To illustrate a routed configuration for which a path cannot be obtained for a specific net N, consider the configuration shown in Figure 7.8a.

The conflict graph H_3 for D_3 is shown in Figure 7.8b. The set V for this example is $\{A^+, B^-\}$. Nodes A and B are in the same component of H_3, they are oppositely labeled, and they are both in V. Thus, a non-switching path does not exist for net N in D_3.

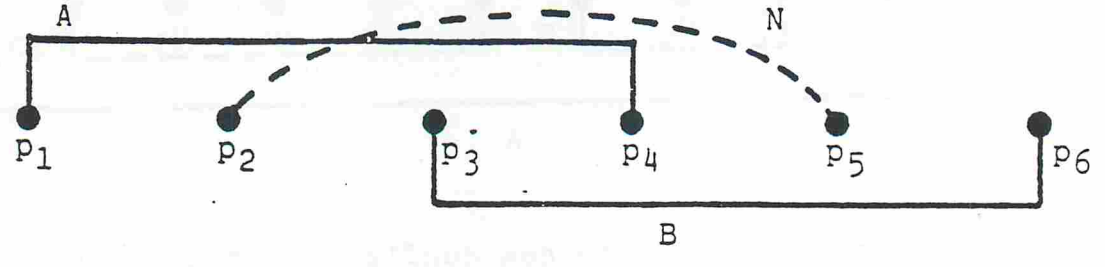


Figure 7.8a. Routed configuration D_3 used to illustrate a routing failure for net N = (p_2, p_5).

Figure 7.8b. The conflict graph H_3 for the configuration D_3.

7.3.4 Timing analysis of Algorithm 7.2

There are no loops in Algorithm 7.2. Therefore, the timing analysis of the algorithm consists of the sum of the number of operations required by each step. In step 1, if the conflict graph is not available, then $O(n^2)$ time is required to generate H as shown in Section 4.2, where n is the number of points in D. If H is available (say from a previous execution of Algorithm 7.2), then step 1 requires $O(n)$ time to generate H'.

Step 2 requires the determination of components of H. Whichever method is used, at least each edge in H must be examined. Thus, any process of complexity $O(k)$ fulfills this requirement, where k is the number of edges in H. It is suggested that the depth-first search method be used

although any spanning tree method of complexity $O(k)$ may be used. The depth-first search method is of complexity $O(k)$ and is simple both conceptually and in implementation. Since $k = n^2/16$ in the worst case (see Section 7.2.3) the complexity of step 2 is $O(n^2)$.

Step 3 requires only linear time since the partitioning of graph components is at most a single examination of each node in H. In step 4, no more than n/4 node labels will change. Thus step 3 and step 4 both operate in time $O(n)$.

From the foregoing discussion, it is seen that the non-switching unidirectional path algorithm is of complexity $O(n^2)$ in the worst-case.

To obtain a reasonable best-case time complexity for the non-switching unidirectional path algorithm, a routing configuration is desired such that the number of arcs k in H be minimum. For $k = 0$, step 2 is of complexity $O(n)$ and step 3 is not even executed. Thus, in the best case, the single-layer unidirectional routing change algorithm is of time complexity $O(n)$.

CHAPTER 8

A CELLULAR APPROACH TO UNIDIRECTIONAL ROUTING

8.0 Introduction

Using an extension of the well-known shortest path technique described by Lee [6], it is possible to apply this technique to find suitable paths in a unidirectional configuration. The technique described in this chapter applies only to an unconstrained configuration. Application of Lee's method to finding minimum length paths in constrained configurations is left for further study.

The basic Lee algorithm divides the two-dimensional routed configuration (not necessarily unidirectional) into a grid of cells with a fixed number of rows and columns. Then a cell marking procedure is performed. Finally, a traceback from the terminal point of the net to be routed to the starting point of the net is performed to find the desired path. The process described in this chapter divides the unidirectional configuration into differing size cells as shown in Figure 8.1 rather than fixed size cells as for the Lee method. The rest of the procedure is identical to that described by Lee.

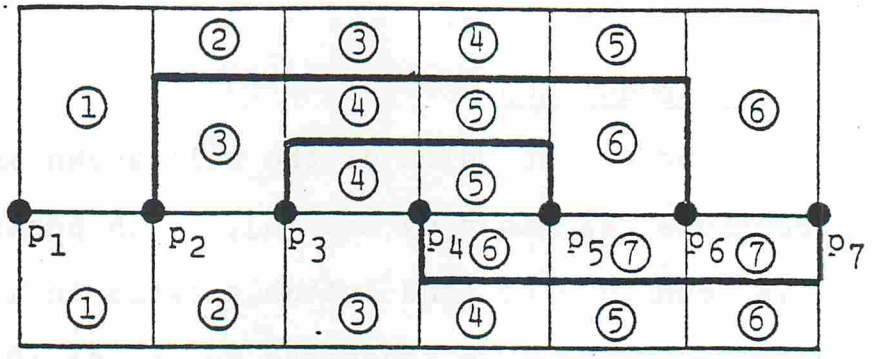


Figure 8.1. An example of the cell structure imposed on a configuration. (The circled integers are the cell values inserted by Algorithm 8.1 assuming a net N = (p_1,p_7) is to be routed.)

8.1 A Minimum-Length Path Cellular Algorithm

The procedure given in Algorithm 8.1 is basically Lee's algorithm with non-fixed size cells.

Algorithm 8.1: Cellular minimum-length path in a unidirectional configuration.

Purpose: This procedure finds the shortest feasible path in a routed configuration D for a two-point net N between two points s and t in D.

Method: Lee's shortest-path procedure using the cell structure as demonstrated in Figure 8.1.

Input: A routed configuration D and the two end points of the net to be routed.

Output: A shortest feasible path for the net in D.

Procedure:

Step 1. Layout the rectangular cell structure on D where upper and lower sides of cells are either the horizontal sections of all paths routed on D, the point line, or the outer track adjacent to the outermost routed path on either side of the unidirectional points. The left and right sides of each cell are imaginary vertical lines intersecting each point in D and extending to the outermost horizontal lines.

Step 2. Label each cell with an integer as follows:

(1) label all cells which are adjacent to the uncovered side of point s with the value 1.

(2) select a cell with a label that has at least one unlabeled adjacent cell such that the label of the cell is minimum. Let the label of the cell be i. Label

each adjacent unlabeled cell i+1. If there are no cells with unlabeled adjacent cells, exit procedure since no path for N exists.

(3) if a newly labeled cell is adjacent to an uncovered side of point t, go to step 3. Otherwise, continue labeling cells by returning to substep (2).

Step 3. Beginning with a minimum-labeled cell (called the *current* cell) adjacent to the uncovered side of point t, work back through the cell structure constructing a path P as follows:

(1) let the label in the current cell be i. Select the next cell in P to be a cell adjacent to the current cell with label i-1. In case several adjacent cells with label i-1 exist, select the next cell in P to be the one to the right or left of the current cell.

(2) let the new cell just selected in the previous substep be the current cell. If this cell has label 1 then a complete minimum-length path P between points s and t has been found so exit this procedure. Otherwise go to substep (1).

8.1.1 An example of the use of Algorithm 8.1

As an example of the use of Algorithm 8.1, consider the configuration shown in Figure 8.2, and assume a net between points p_3 and p_9 is to be routed.

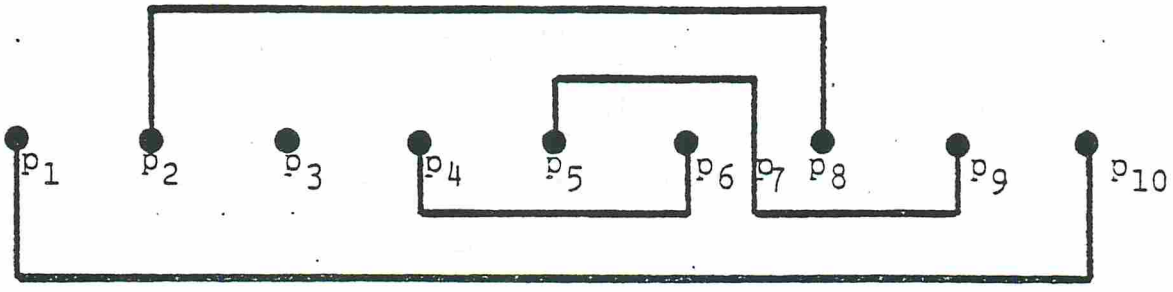


Figure 8.2. A routing configuration illustrating the use of Algorithm 8.1. Net $N = (p_3, p_9)$ is to be routed.

A rectangular cell structure, shown in Figure 8.3, is constructed on the configuration, and the cells are labeled per steps 1 and 2 of Algorithm 8.1. A minimum-length path as found in step 3 is shown as a dotted line.

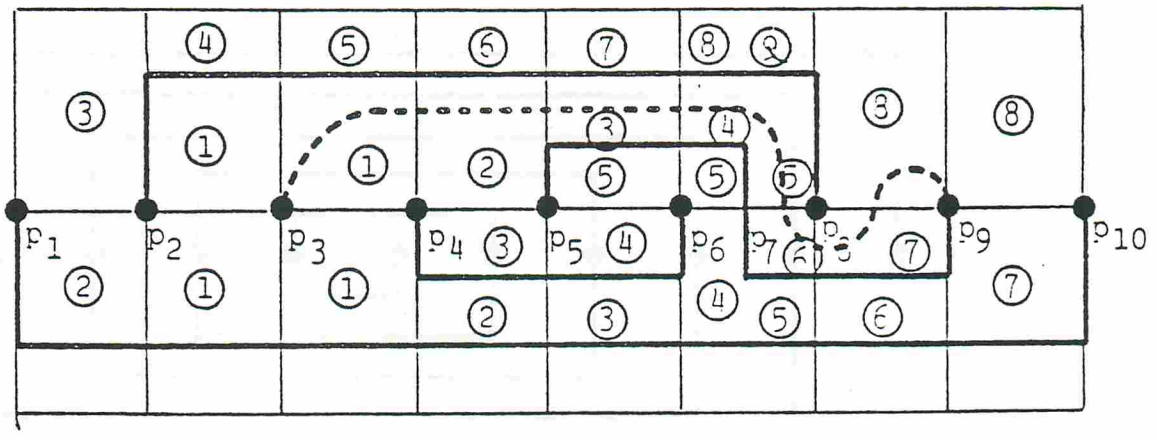


Figure 8.3. The complete labeled cell structure for the configuration shown in Fig. 8.2.

8.1.2 Timing analysis of Algorithm 8.1

Let B be the number of cells created in a configuration D, and let n be the number of points in D. Obviously, the timing complexity of Algorithm 8.1 is dependent upon the number of cells B in D. The minimum number of cells occurs when D has no routed paths. The maximum number of cells is obtained when D has the maximum number of nets routed such that the total length of all sections is maximized. This occurs for n-1 segments routed as shown in Figure 8.4.

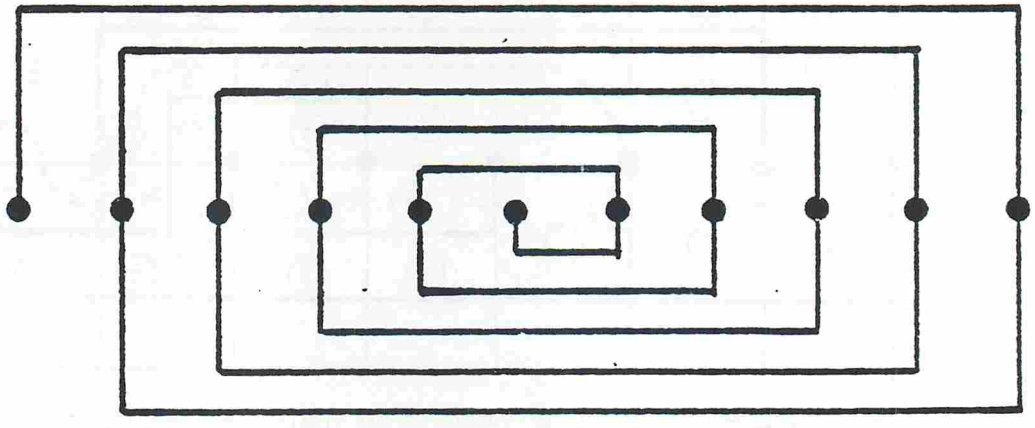


Figure 8.4. An example of a configuration which maximizes the number of cells.

In this case,

$$B_{max} = 1 + 3 + 5 + \cdots + n-1 + 2 + 4 + 6 + \cdots + n-2$$
$$= \frac{1}{2}n(n-1) .$$

In step one, a minimum of 2n and a maximum of $\frac{1}{2}n(n-1)$ cells must be constructed. If we assume each cell requires unit time to layout, then step one requires at most $\frac{1}{2}n(n-1)$ operations. In step two, a minimum of one and a maximum of $\frac{1}{2}n(n-1)$ cells must be labeled. At most three neighbors must be examined for each labeled cell to determine if they have been labeled or not. Thus, step two requires at most $\frac{3}{2}n(n-1)$ operations. Step three requires at least one operation if the source and terminal points are adjacent in D, but it requires at most $\frac{1}{2}n(n-1)$ operations if all of the cells must be retraced. Thus, for the entire algorithm,

$$2n+2 \leq \mathcal{O}ps \leq \frac{5}{2}n(n-1) .$$

8.2 Discussion

It is interesting to compare this method with the graphical method discussed in Chapter 3 for unconstrained configurations. Although the graphical method required only $\mathcal{O}(n\log n)$ operations to find a minimum-length path for a net in a configuration, whereas the extended Lee method described in this chapter requires $\mathcal{O}(n^2)$ in the worst case,

the capabilities of either method are identical. The Lee method has been presented to illustrate both that (1) conventional routing techniques can be used to solve at least the unconstrained unidirectional routing problem for minimum-length paths, and (2) solutions methods tailored to specific problems such as unidirectional routing can sometimes be more efficient than general methods applied to the same problem.

CHAPTER 9

EXPERIMENTAL RESULTS

9.0 Introduction

Whereas theoretical aspects of optimal unidirectional routing have been presented thus far, in this chapter we attempt to present the results of some analysis performed to both validate this research as well as provide further insight into the capabilities of this form of routing. Using simple statistics and a computer program implementing the major algorithms presented in Chapters 2, 3, 4, and 5, we obtain results which clearly show the viability of these unidirectional routing methods when applied to large boards.

The results presented here are generally in terms of averages rather than worst-case values. Thus, timing results are average times obtained from ensembles of data. Similarly, performance parameters, such as execution times and completion rates, are given in terms of average values obtained from synthetically generated problems. No routing problems using real boards have been analyzed.

9.1 The Program

9.1.1 Description

A routing program, called UNIROUT, was written and used to provide all the data summarized in this chapter. Written in FORTRAM IV to run on a CDC 7600 computer, it implements all procedures as presented in Chapters 2, 3, 4, and 5 with the following exceptions:

1. the implementation of Dijkstra's least-cost procedure does not incorporate the more efficient search method suggested in Chapter 2. Rather the standard linear search method is used, and hence this algorithm has a time complexity of $O(n^2)$.

2. Algorithm 4.4 which updates a feasible graph G' to produce a final access graph G^*, is not implemented. Rather, a less efficient but simpler update procedure is used to produce a new configuration directly from the feasible access graph.

3. Algorithms 5.2 and 5.3 which assign minimum-perturbation paths to tracks in a configuration are not implemented. Rather, the unconstrained case update procedure is used to assign tracks to arcs for the floating-track case. Processing from the point line outward

in both streets, this procedure assigns all segments to free tracks nearest the point line.

The UNIROUT program is organized to first read in a problem set, and then route nets as directed by the control card in the problem set. The problem set consists of a control card, a configuration description, and a net list. Subroutine INPUT reads all input, syntactically analyzes it, and constructs the data structure used throughout the rest of UNIROUT.

After the input has been completely read into the data structure, each net in the net list is routed, one after the other, by repeatedly executing subroutines GENER8 which creates the access graph, MIN which finds a minimum-cost path in the access graph, ASSIGN which assigns segments to tracks, and UPDATE which updates the program data structure in preparation for routing the next net.

9.1.2 Input

Two sets of inputs were prepared and processed. The first set (called INPUT 1) consists of 24 cases composed of eight groups of three problem sets each. Each problem set consists of an empty configuration and a net list. The eight groups correspond to problem sizes of 10 point, 20 points,..., and 80 points in the configurations. The number of nets to be routed per problem set varies from 5 to

21 but it never exceeds half the number of points in the configuration.

All 24 cases were processed by UNIROUT for each *configuration type* (namely unconstrained, fixed-track, and floating-track) and each *cost goal* (namely, minimum-length, minimum-congestion, and minimum-perturbation).

When a problem set is executed by UNIROUT, it is identified by a two-character label and a two-digit number. The first character is either U, X, or F, and identifies the problem set configuration type (see Table 9.1). The second character is either L, D, or P, and identifies the routing cost goal to be used in routing the nets contained in the net list. Table 9.1 defines these characteristics. The first digit indicates to which group the problem set belongs (e.g., "1" for the 10-points per configuration group, "2" for the 20-points per configuration group, etc.), and the second digit identifies one of the three problem sets in the group. For example, problem set UL-3.2 is executed with an unconstrained configuration with minimum-length routing cost goal. The configuration has thirty original points, and the problem set is the second one in the 30-points per configuration group.

The net lists for all problem sets in INPUT 1 were randomly generated by computer. The net length distribution is weighted towards shorter nets in an attempt to

simulate what would be reasonably expected in realistic routing situations. (The length of a net is the distance between the end points of the net.) An example of the net length distribution for 138 nets which individually could potentially be routed in a configuration of 32 points is shown in Figure 9.1.

Table 9.1. Problem Set Designators

	Designator	Meaning
Configuration Type	U	Unconstrained configuration
	X	Fixed-track configuration
	F	Floating-track configuration
Routing Cost Goal	L	Minimum-length path
	D	Minimum-congestion (i.e., "density") path
	P	Minimum-perturbation path
Configuration Size	1	20 points, 10 segments (1/2 of points)
	2	50 points, 25 segments (1/2 of points)
	3	80 points, 20 segments (1/4 of points)
	4	80 points, 40 segments (1/2 of points)
	5	80 points, 60 segments 3/4 of points)

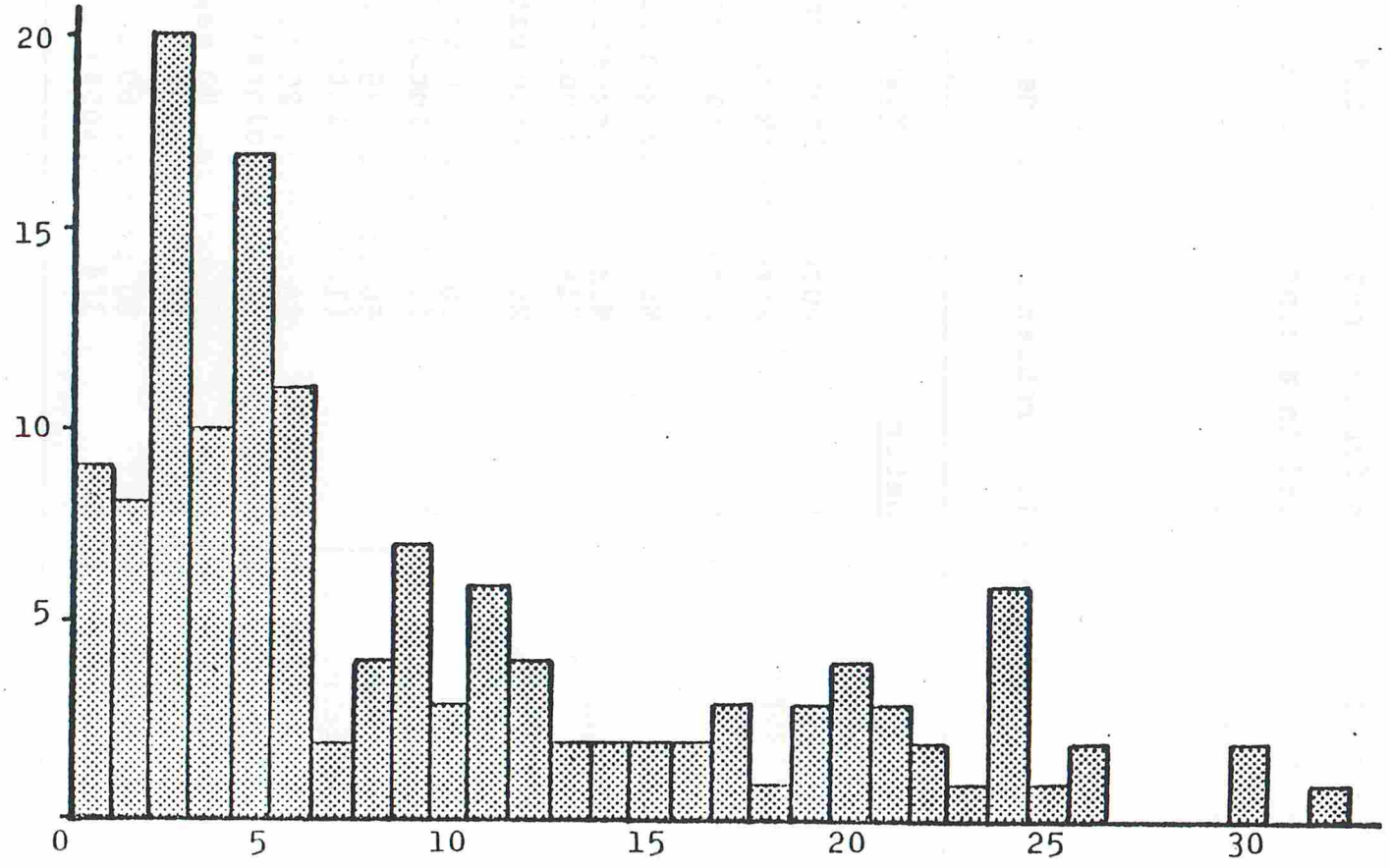


Figure 9.1. Length distribution of 132 nets on 32 points.

The output from INPUT 1 was used to obtain an assessment of routing completion rates discussed in Section 9.4. This same output was also used to prepare the original configurations for INPUT 2, the second of the two sets of input data.

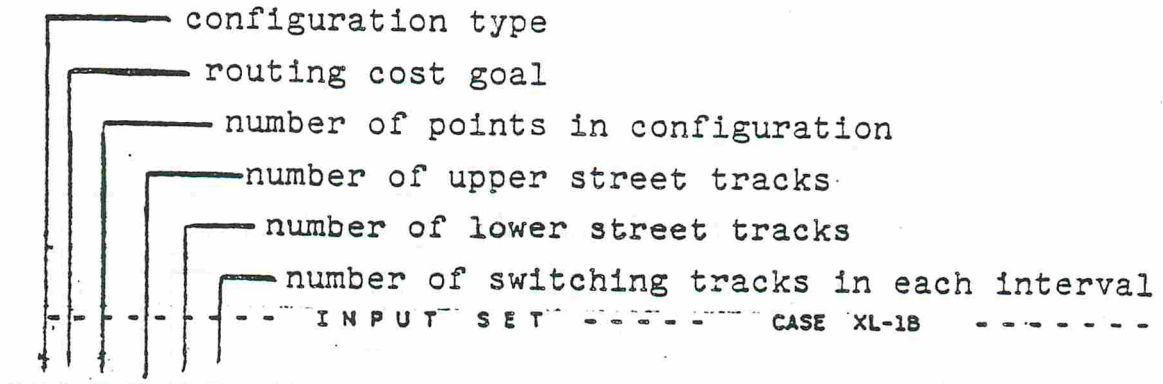
The second input set (called INPUT 2) consists of 15 cases composed of five groups of three problem sets each. Each problem set consists of a non-empty configuration and a net list. The groups are identified by the number of points and segments in the configurations within the group. Table 9.1 shows the five configuration sizes considered in INPUT 2. When an INPUT 2 problem set is executed by UNIROUT, it is identified by a label similar to INPUT 1 problem sets. Rather than a two-digit part of the label, however, INPUT 2 problem sets use a digit code to refer to the configuration size, and a letter designator A, B, or C to identify one of the three problem sets in the group. For example, problem set UL-1A is executed as an unconstrained configuration with minimum wire length as the routing cost goal. The configuration has ten points and five routed segments, and it is the first of three problem sets in the 10-points per configuration group.

Each problem set in INPUT 2 contains a random list of 30 nets with the exception of 50-point, 25-segment problem sets each of which contain only 20 nets. All problem sets

of a given configuration size and suffix designator (i.e., A, B, or C) have identical net lists. For example, the net list for problem set XD-3A is the same as for FP-3A.

INPUT 2 is used to obtain execution times for each of the major subprograms of UNIROUT as a function of the number and type of arcs in a path in an access graph. Whereas each problem set in INPUT 1 was executed in the standard way to route each net in a configuration which includes all previously routed nets in the net list, each net in INPUT 2 was routed only in the original configuration in the problem set. This permits a controlled investigation of the performance of the program.

As an example of these inputs, note the input listing in Figure 9.2. The character in column 1 of each card identifies the card type. The "C" card is the control card. The "T" cards provide titling information for output plots. The "S" card prevents the use of the updated configuration for routing the next net. Instead, each net in the net list is routed in the original configuration. The "P" card identifies the switching track locations occupied by the points in the configuration. The "U" and "L" cards identify routed segments in the upper and lower streets respectively. The "N" cards are the net list. The configuration described by the P, U, and L cards of Figure 9.2 is shown in Figure 9.3.




```
C X L 20 10 10 3    XL-1B
T FIXED-TRACK, MINIMUM-LENGTH   20 POINTS, 10 SEGMENTS
T CASE  XL-1B    10 UPPER TRACKS, 10 LOWER TRACKS, 3 SWITCHING TRACKS
P 1 5 9 13 17 21 25 29 33 37 41 45 49 53 57 61 65 69 73 77
S
U  1  17  10
U  4  16   7
U  6  10   4
U  7   9   1
U 13  14   1
L  4  20  10
L  6  19   7
L  8   9   1
L 12  18   4
L 15  16   1
N 19  20
N 15  20
N  2   3
N  7  11
N 11  19
N  5  18
N  7  10
N 10  13
N  7  12
N  2   5
N  2  12
N  1  11
N  3  11
N 10  15
N  5  20
N 18  19
N  5  15
N 12  15
N 11  13
N  3  19
N 10  14
N 13  17
N  3  10
N  8  14
N  5   7
N  5  13
N  7   8
N  2  18
N  3  15
N  3   7
E
```

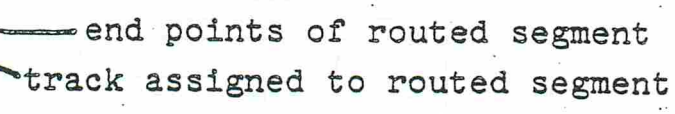


Figure 9.2. Input listing for a fixed-track minimum-length configuration.

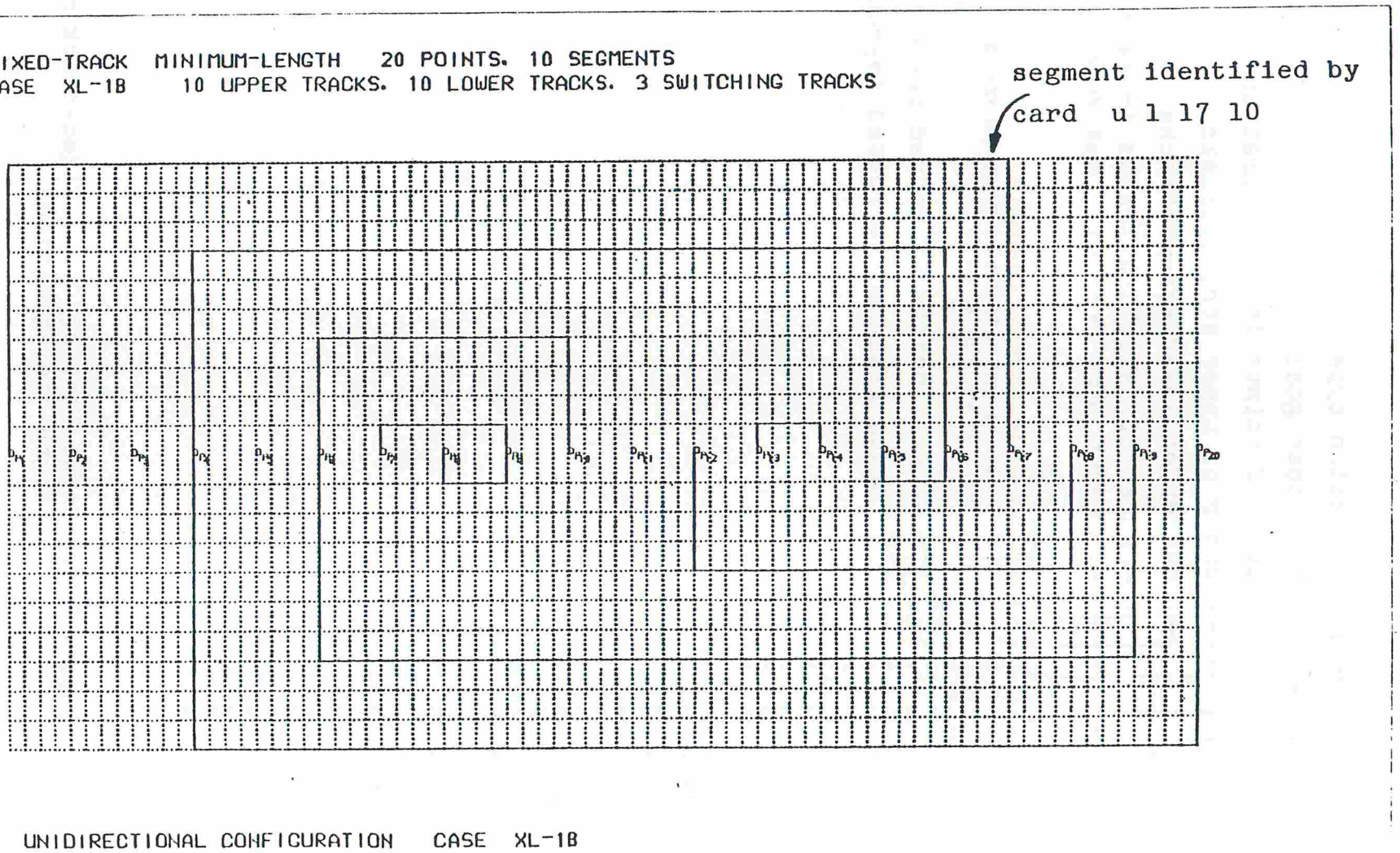


Figure 9.3. The input configuration described by the input listing in Fig. 9.2.

9.1.3 Output

Three forms of output result from executing any of the inputs just described. An output listing giving pertinent data for each net routed is provided. Each line of the listing contains information for a routed net such as the net end points, the number of nodes and arcs in the access graph *after* the net is routed, the number of arcs in the path for the net in the access graph *before* the net is routed, and execution times for each major subprogram in UNIROUT.

A second form of output is punched cards where each card contains data similar to that presented in the output listing. An optional plot of the configuration represented by the input either before or after any net has been routed is also output.

9.2 Performance

9.2.1 General results

The results obtained from executing INPUT 1 are summarized in Table 9.2. As expected, all unconstrained and most floating-track routing problems achieved 100% routing. However, the fixed-track problems achieved mixed results.

There was only one problem set for which the floating-track configuration had a routing failure. The reason for this can be seen in Figure 9.4 where a pathological case exists. Note that point p_1 is connected to point p_{11} by

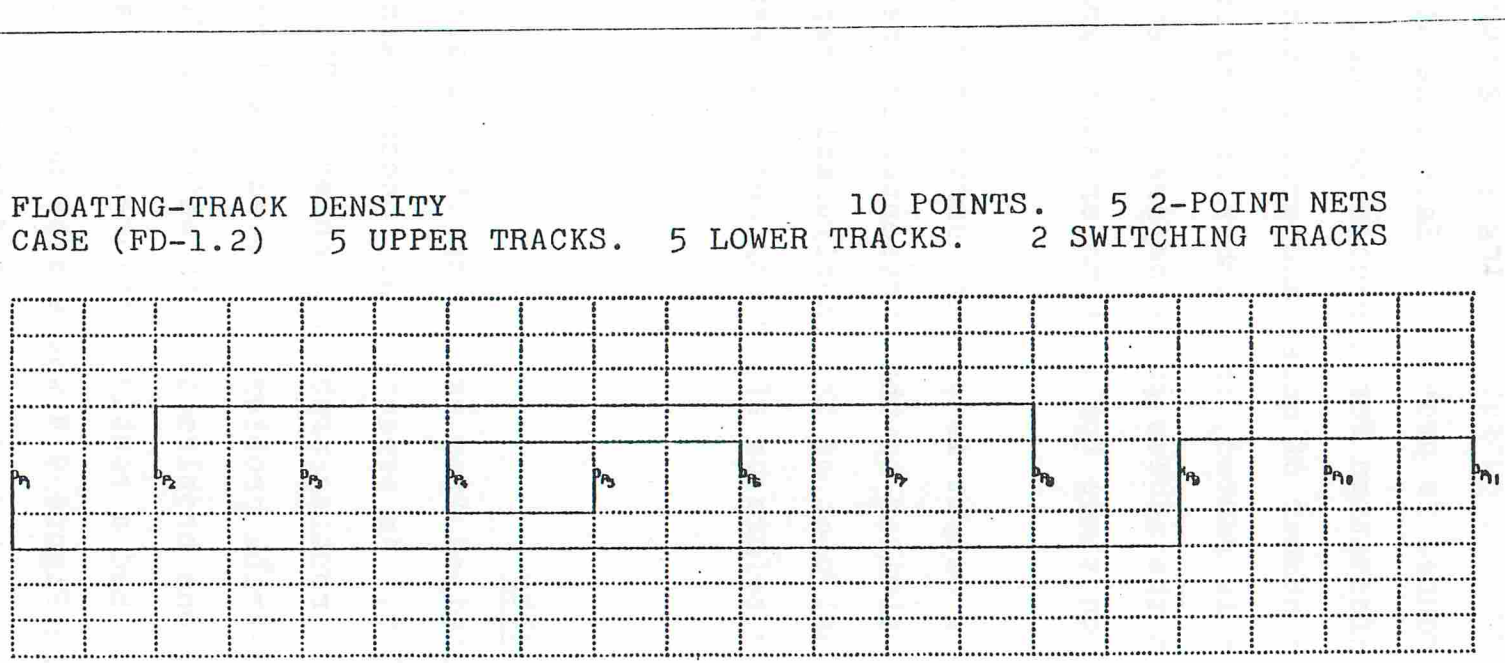


Figure 9.4. A configuration illustrating the failure of a net to be routed.

two nets which have point p_9 in common. Thus, the route for net (3-10) is blocked and no path exists for the net. In general, if the first and last points in a configuration are connected by at least two or more segments, then all subsequent nets with end points lying on each side of the path connecting the first and last points cannot be routed. One way to prevent this from occurring is to route nets which contain either the first or last points in the configuration after all other nets in the net list are routed.

Table 9.2. Routing Completion for INPUT 1 Problem Sets.

	Number Nets Attempted	Number Nets Routed	Percent Complete
Unconstrained - Min Length	317	317	100
Fixed Track - Min Length	317	280	88.3
Min Congestion	268	209	78.0
Floating Track - Min Length	317	317	100
Min Congestion	317	316	99.7
Min Perturbation	317	317	100

9.2.2 Fixed-track performance

Table 9.2 indicates lower completion rates for fixed-track routing problems compared with the other cases. There are several reasons for this.

1. The algorithms used do not necessarily globally optimize track assignments.

2. No sorting of the net list was accomplished prior to executing the routing program.

Fixed-track routing requires further study. A brief examination of the data shows that some heuristic "rules" may be easily incorporated that could improve the completion rate. For instance, a pass over the net list could find groups of nets that occupy local regions of the configuration and sort the list accordingly. Furthermore, nets from the same multipoint net could be made as short as possible by ordering the nodes of the multipoint net smallest to largest and then route each resulting two-point net in the same order. For example, the multipoint net (5,3,8,2) would appear in a net list as {(5,3),(3,8),(8,2)} if it were not sorted.[1] However, with the net list sorted as described, the more straightforward configuration in Figure 9.6 results. It is seen in Figure 9.5 the density in the streets between p_4 and p_5 is 3 which leaves only 20-3 = 17 tracks available for future nets, while for Figure 9.6 19 tracks are available. For this study we did not order the net list since we did not want our results to be dependent on other parameters such as ordering. The resulting routed paths appear in Figure 9.5.

[1] In all cases investigated here maximum "randomness" is desired.

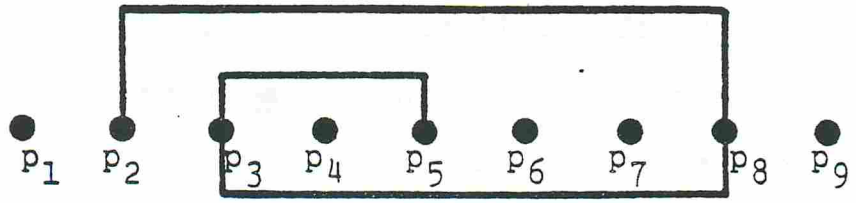


Figure 9.5. A routed configuration resulting from the net list {(5,3),(3,8),(8,2)}.

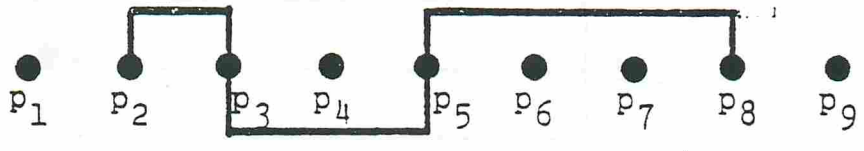


Figure 9.6. A routed configuration resulting from the net list {(2,3),(3,5),(5,8)}.

Finally, it was noticed that once a configuration was routed with enough nets to cause a net to fail to be routed, the tendancy was for all subsequent nets in the net list to also fail to be routed. Figure 9.7 graphically illustrates this point. A ● represents a failed net and | represents the last net in the net list. The results shown here are consistent with the findings of Agrawal [29] relating routing completion to routing density, namely once a critical routing density is reached, most future nets fail to be routed.

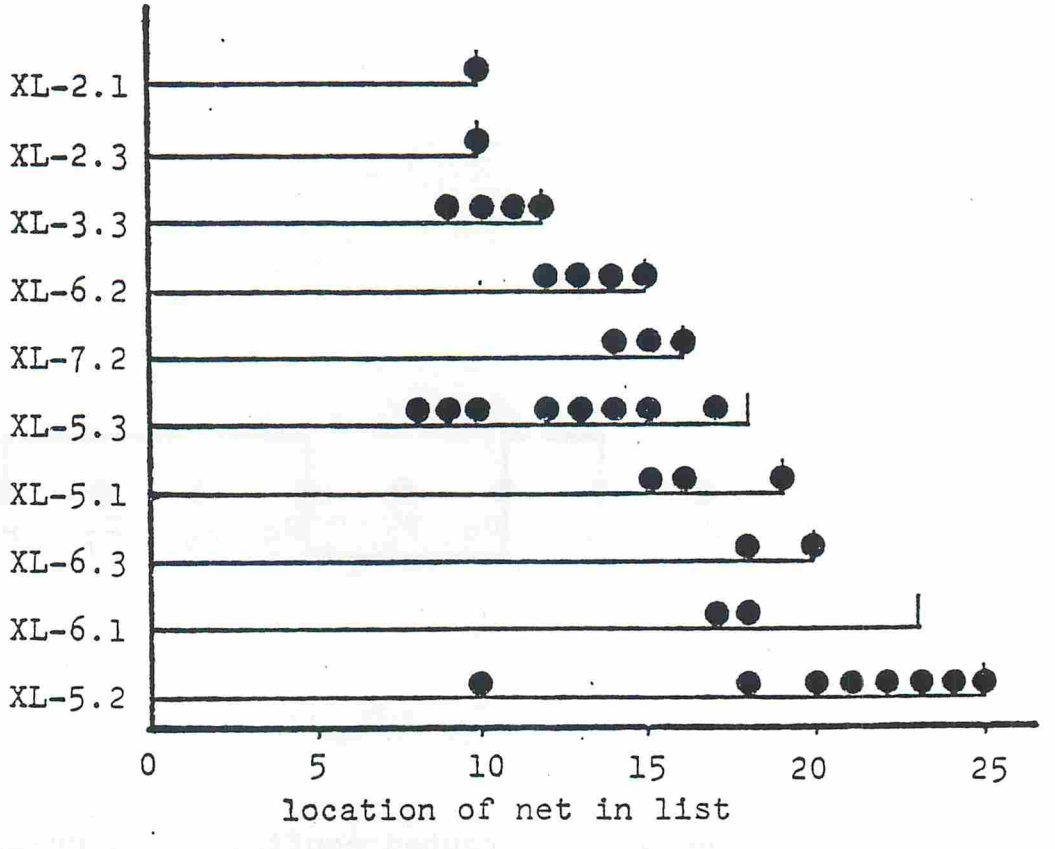


Figure 9.7. Plot of failed nets.

9.3 Execution Times

9.3.1 The analysis

The output from the routing program using INPUT 2 data was analyzed to determine the more significant characteristics concerning the execution times of UNIROUT.

The execution times output from the three problem sets in each group were plotted as a function of both the total number of arcs in the path and the access graph, and the number of switching arcs in the same path. Those paths which required excessive execution times (due to plot I/O times) or which were not routed were discarded. Linear to 5^{th} order least-squares regression lines were obtained from the remaining data. Figure 9.8 shows a plot, in this case the total execution time[2] versus path length (number of arcs in the path in the access graph) for case UL-2 which has 50 points in the configuration. The isolated triangle symbols represent raw data, the broken line connects average times for each path length, and the smooth curve is the quadratic regression line for the raw data.

9.3.2 Access graph construction

The access graph construction subprogram (called GENER8 in the routing program) consists of three graph construc-

[2]Total execution time is the time to generate an access graph, find a minimum-cost path, and update the configuration with the new net. For unconstrained problem sets, track assignment is not performed.

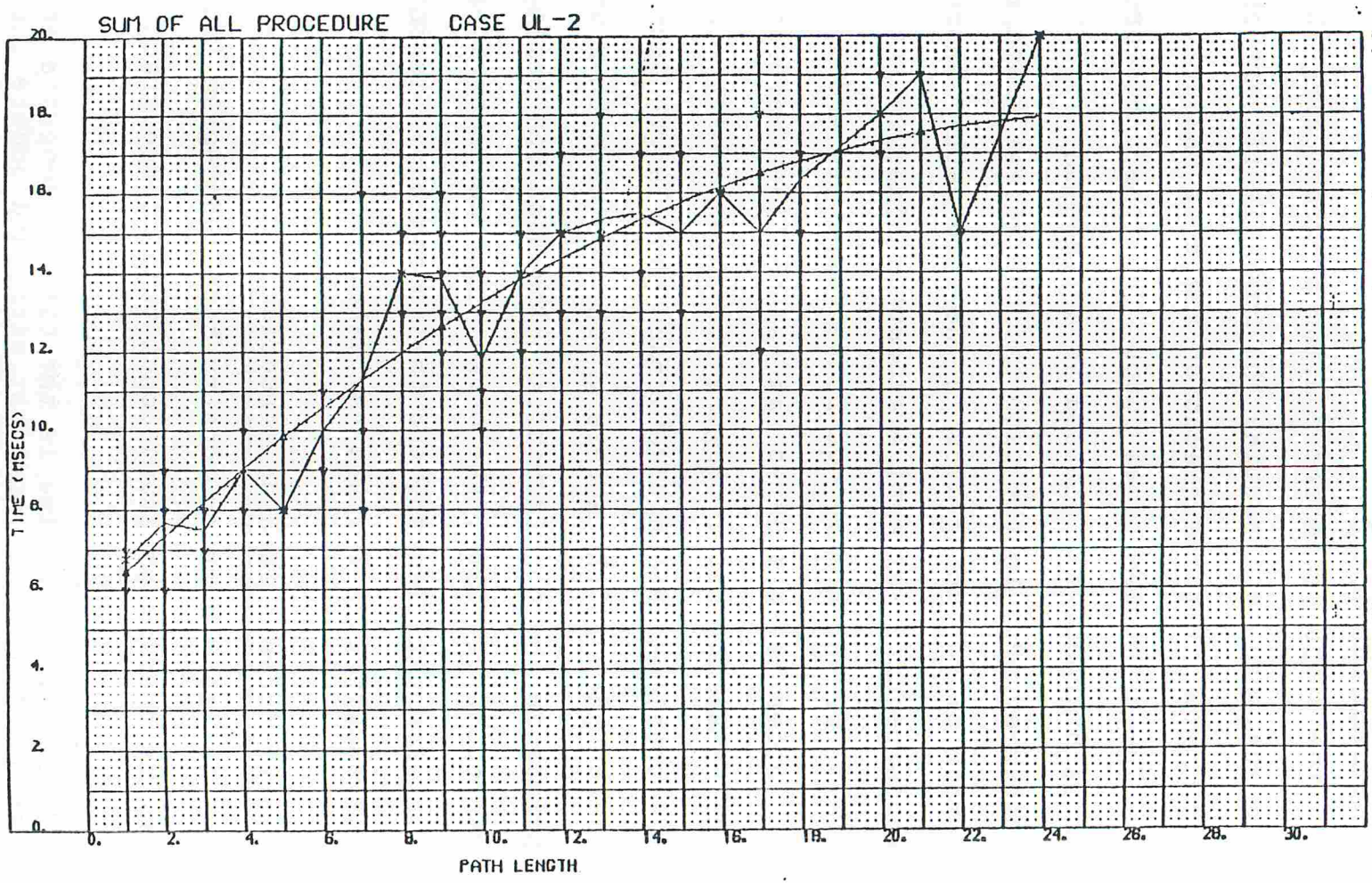


Figure 9.8. A typical example of an analysis output plot.

tion subroutines one each for the unconstrained, fixed-track, and floating-track configurations. The execution times for these routines are summarized in Figure 9.9. The configuration density is a measure of the density of a configuration and is defined to be the ratio of the number of routed segments to the number of points in the configuration. Note that the time requirs to construct the access graph is a linear function of both configuration size (Figure 9.9a) and configuration density (Figure 9.9b) as predicted by the theoretical timing analysis presented in Chapters 3, 4, and 5. For the floating-track access graph, the expected worst-case complexity of $O(n^2)$ (where n is the number of points in the configuration) was not observed. The reason the floating-track access graph algorithm takes slightly longer execution time than for the other two types of access graphs is because as the algorithm executes, the stack must be scanned for each arc in the graph to determine the number of tracks that would potentially have to move if the new path contained that arc. The stack is not scanned in either of the other access graph generation algorithms.

9.3.3 Minimum-path timing analysis

9.3.3.1 Minimum-length paths

The execution times required to find a minimum-length path in an access graph are shown in Figure 9.10 as a func-

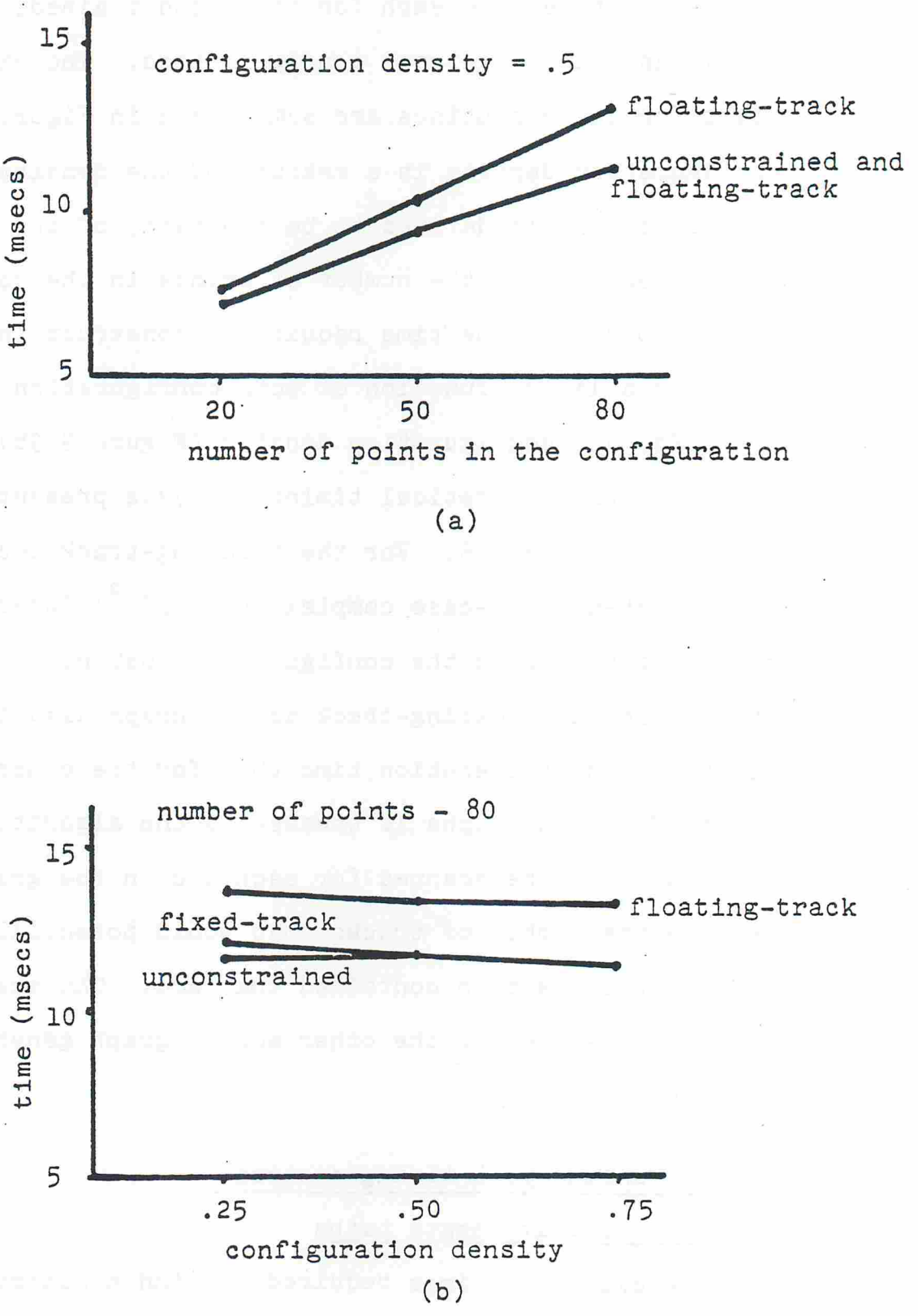


Figure 9.9. Execution times for graph construction.

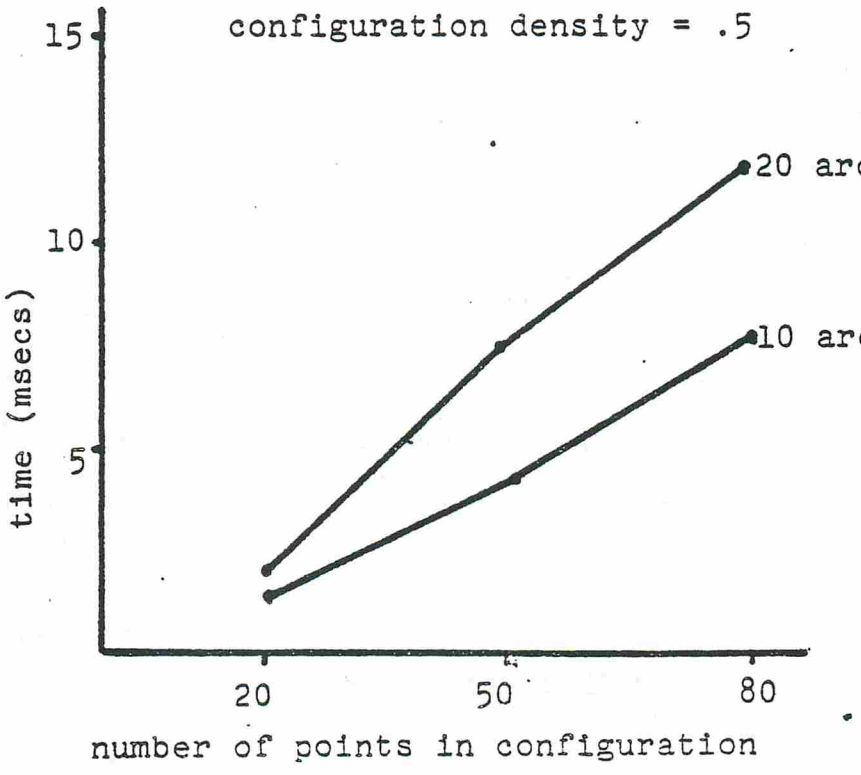


Figure 9.10. Execution times for the minimum-length path procedure.

tion of the number of points in the configuration represented by the graph. Note that as the number of points in a configuration increases, the execution time to find a minimum-length path grows approximately linearly. The same relation holds for the number of nodes in the access graph since the number of nodes is exactly twice the number of points in the configuration.

It was also observed that there is no significant impact on execution time due to configuration density or configuration type.

9.3.3.2 Minimum-congestion paths

Execution times are shown in Figure 9.11 for the minimum-congestion path procedure as a function of the number of points in the configuration represented by the graph in which the paths are to be found. Longer paths show a decidedly non-linear trend because the $O(n^2)$ nature of the minimum-cost algorithm has a greater effect for larger n.

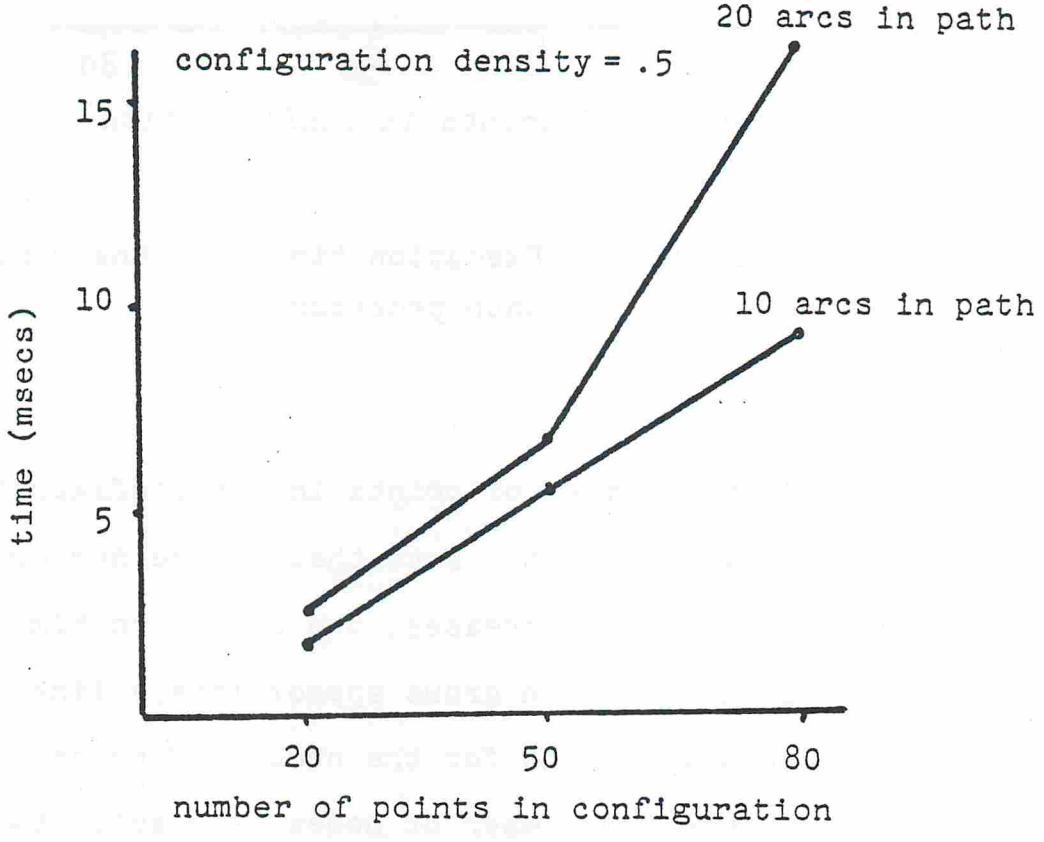


Figure 9.11. Execution times for the minimum-congestion path procedure.

9.3.3.3 Minimum-perturbation paths

The execution times required to find minimum-perturbation paths in an access graph are shown in Figure 9.12 as a function of the number of points in the configuration represented by the graph.

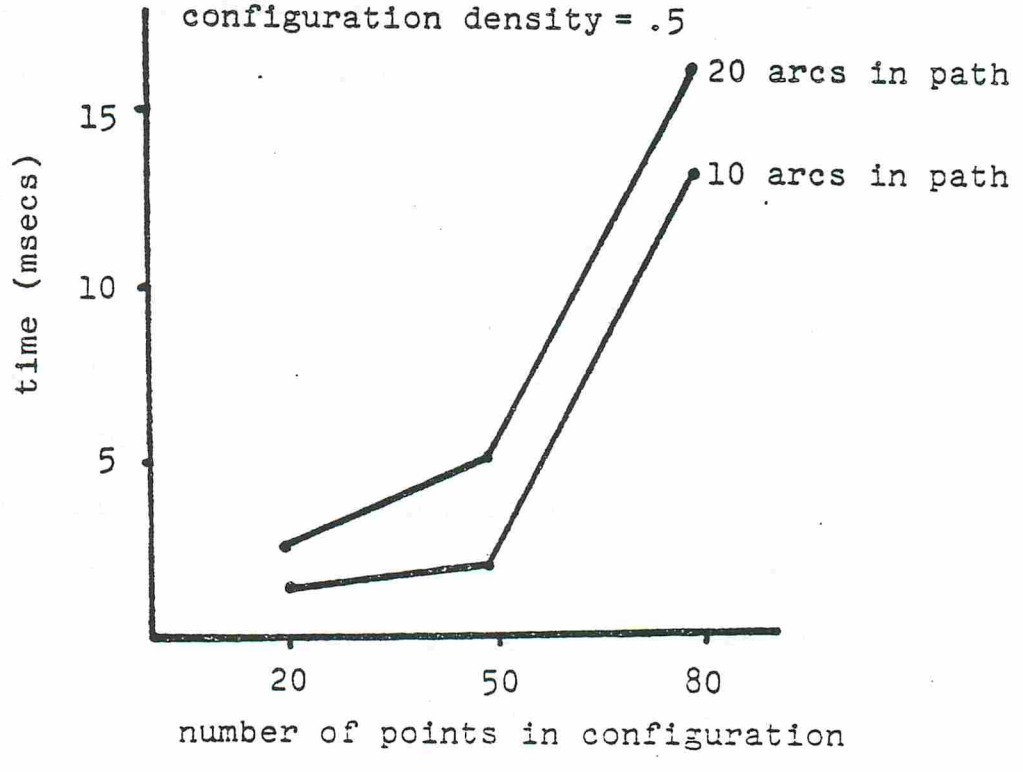


Figure 9.12. Execution times for the minimum-perturbation path procedure.

It is apparent from Figure 9.12 that as the number of points in the configuration increases, the time to find a minimum-perturbation paths grows at least quadratically

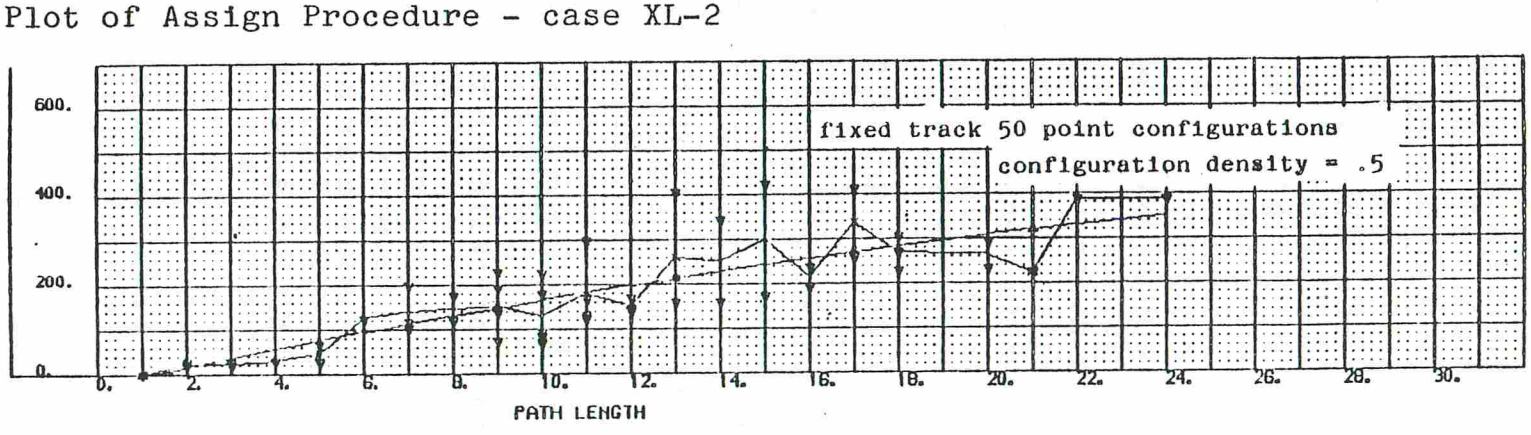
with the number of points. It is apparent that for both minimum-perturbation and minimum-congestion routing, the time-saving search technique suggested in Chapter 2 for the Dijkstra algorithm would be useful in reducing the execution time growth as the number of points in the configuration increases.

As for both the minimum-length and minimum-congestion procedures, there was no significant impact on execution time due to configuration density or configuration type for the minimum-perturbation procedure.

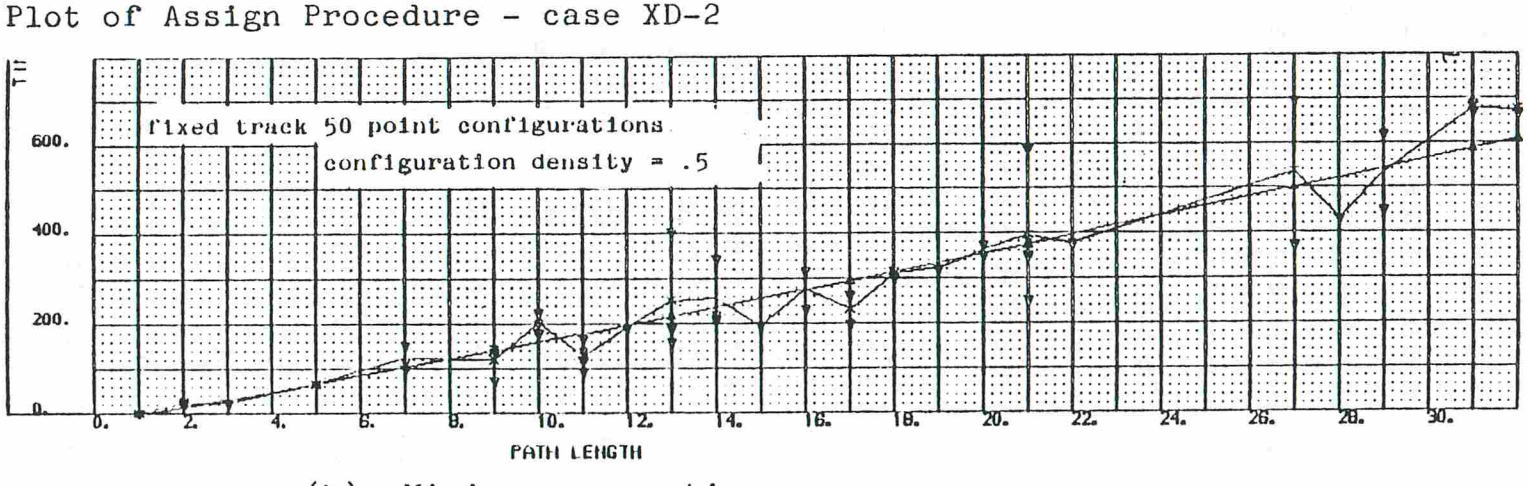
9.3.4 Track assignment analysis

Since track assignment is not needed for the unconstrained case (there are no tracks!), and track assignment for the floating-track case was implemented as part of the update procedure (the update procedure prepares the configuration for routing the next net in the net list), only fixed-track assignment timing will be discussed here.

Figure 9.13 shows two examples of the execution times required to assign paths to tracks as a function of the number of arcs in each path. The results for minimum-length path track assignment shown in Figure 13a shows a well behaved spread of points and an approximately linear relationship between the track assignment procedure execution times and the number of arcs in the path being assigned to tracks. The spread of points is primarily due to the



(a) Minimum-length



(b) Minimum-congestion

Figure 9.13. Execution times to assign a feasible path to tracks in fixed-track configurations as a function of the number of arcs in each path.

variation in the number of segments in the path.

The results for minimum-congestion track assignment shown in Figure 9.13b are similar to those for the minimum-length assignment results. Note that the number of arcs in minimum-perturbation paths tend to be larger than in minimum-length paths.

It was noticed that the number of switching arcs in each path was correlated to some degree with the total number of arcs in each path. A plot of the number of switching arcs in a path versus the total number of arcs in the path for the fixed-track, 50-point configuration case is shown in Figure 9.14. This plot supports the observation. Thus, the same results for execution times as a function of the total number of arcs in a path also tend to hold as a function of the number of switching arcs in the path. Since the number of segments in a new path in a configuration is one more than the number of switches in the path in the access graph representing the configuration, the execution time results given in this section also apply as a function of the number of segments in a new path in a configuration.

An analysis was made of the execution time to assign paths to tracks both as a function of the number of points in a configuration and of the configuration density. Figure 9.15 shows the plots for minimum-length paths, and Figure 9.16 shows the plots for minimum-congestion paths.

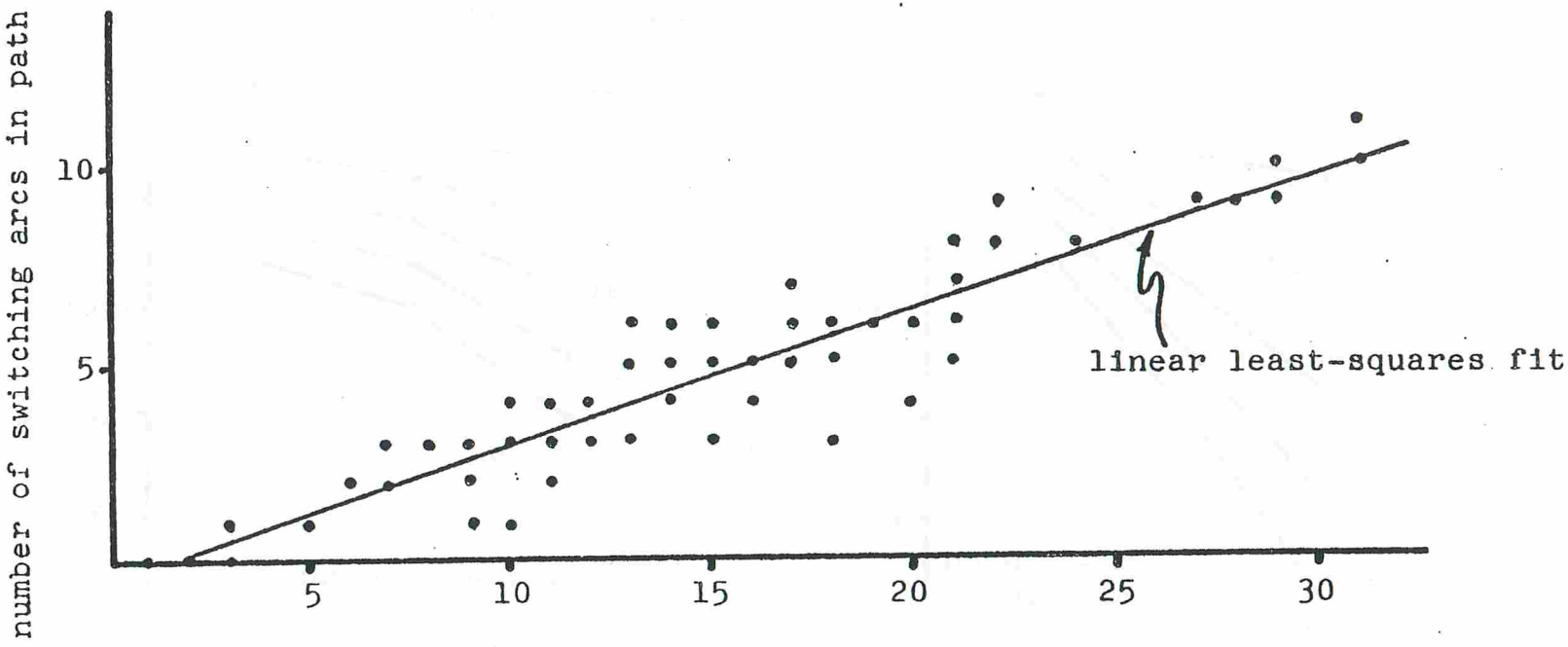


Figure 9.14. Plot of number of switching arcs versus total number of arcs in a path.

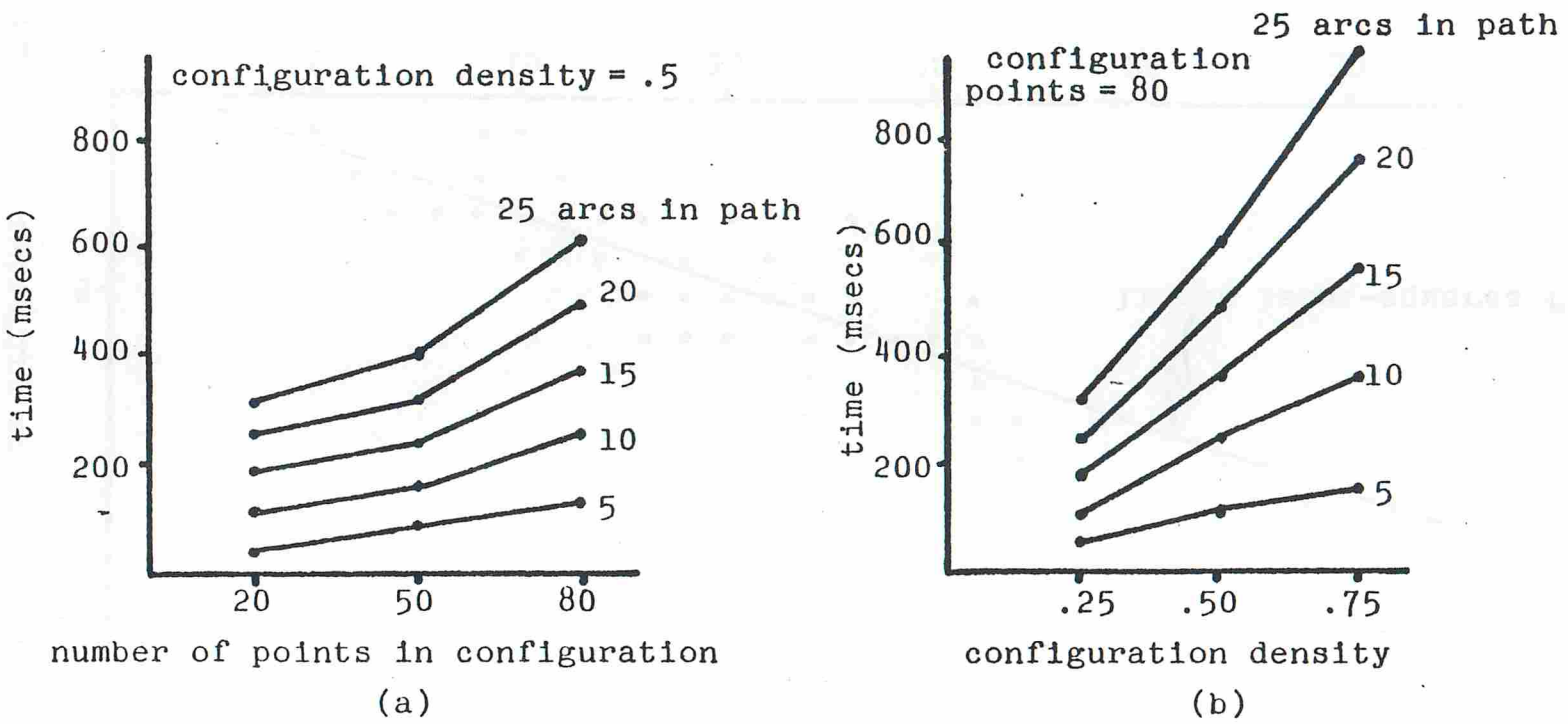


Figure 9.15. Execution times to assign minimum-length feasible paths to tracks in fixed-track configurations.

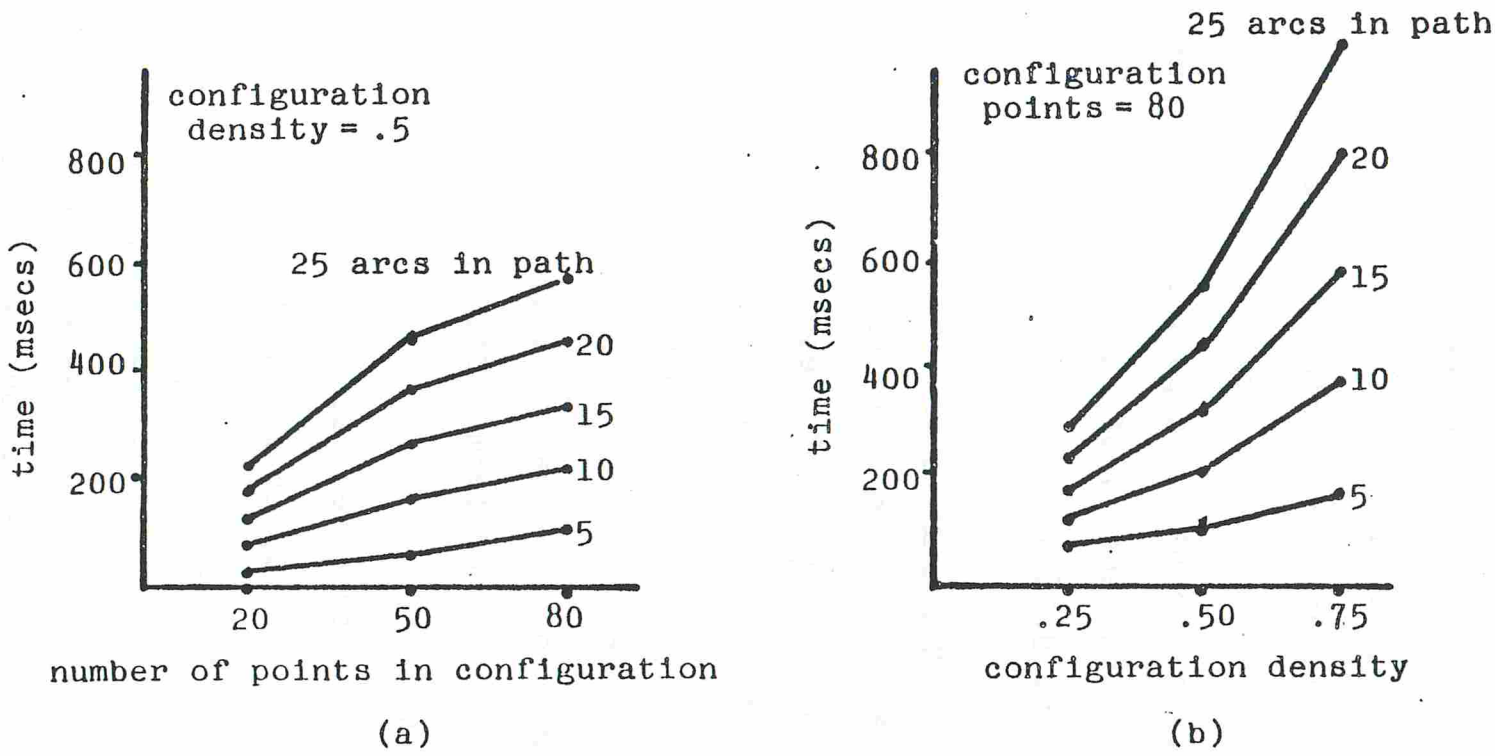


Figure 9.16. Execution times to assign minimum-congestion feasible paths to track in fixed-track configurations.

The results of this analysis show that as the number of points in a configuration increases, the time to assign minimum-length paths to tracks increases as a function of n^{α} where $\alpha > 1$ and n is the number of points in a configuration (Figure 9.15a), minimum-congestion paths increases as well but at a linear rate (Figure 9.16a). However, both path types exhibit a polynomial increase in execution time as the density of the configuration increases (Figures 9.15b and 9.16b).

9.3.5 Update analysis

The two update routiens used in UNIROUT are UPDATEU and UPDATEX. Both of these routines construct a new configuration given an old configuration, the feasible path, and, in the case of UPDATEX, the tracks assigned to new segments and switches. UPDATEU is used for both the unconstrained and floating-track problems, whereas UPDATEX, which is derived from UPDATEU, is used for fixed-track problems.

They both essentially run in times which increase linearly and slowly as a function of the number of arcs in a path and the number of points in a configuration. Furthermore, the update routines make only a minor contribution to the total execution times for UNIROUT. For example, whereas the total time to route a net in an unconstrianed 80-

Table 9.3. Summary of Experimental Results

Process	Function of Path Arcs		Function of Points		Function of Density	
	observed	expected	observed (average)	expected (worst-case)	observed	expected
Access Graph Construction						
Unconstrained			linear	linear	linear	—
Fixed-track			linear	linear	linear	—
Floating-track			linear	quadratic	linear	—
Minimum-path Algorithm						
Length	linear	—	quadratic*	quadratic*	—	—
Congestion	linear	—	quadratic*	quadratic*	—	—
Perturbation	linear	—	quadratic*	quadratic*	—	—
Assignment						
Unconstrained	—	—	—	—	—	—
Fixed-track	linear	—	quadratic	quadratic	quadratic	—
Floating-track	—	—	—	linear	—	—
Update						
Unconstrained	linear	—	linear	—	linear	—
Fixed-track	linear	—	linear	—	linear	—
Floating-track	linear	—	linear	—	linear	—

Note: *$\mathcal{O}$(nlogn) search function not implemented.

point configuration with density = .5 was 31 msecs, UPDATEU only took 8 msecs. For UPDATEX, execution times never exceeded 2% of the total time necessary to route a net.

9.4 Conclusions

The experimental results presented in this chapter support the theoretical timing analysis presented in earlier chapters. Table 9.3 summarizes these results.

CHAPTER 10

CONCLUSION

10.0 Summary

In this work we have studied the problem of unidirectional routing, an important problem in routing boards with rectilinear point spacing and fixed-vias. Algorithms have been proposed for routing between unidirectional points such that routing is locally optimal with respect to minimum wire length, minimum board congestion, and minimum routed wire movement. Furthermore, each of these routed wiring goals has been examined in light of three types of unidirectional configurations: unconstrained board configuration where neither tracks nor street boundaries exist, fixed-track board configurations where once segments of a path are assigned to tracks they are never reassigned to accommodate the routing of subsequent paths, and floating-track board configurations where wires may be reassigned to free tracks to accommodate the routing of subsequent paths.

Several special cases have been examined. A unique graphical method using bipartite graphs has been presented which routes a minimum-switch path in a unidirectional configuration. In addition, the special case of routing a

path in a unidirectional configuration where no switches are permitted was solved using the concept of bichromatic graphs. Finally, using non-standard cells, the Lee algorithm was adapted to solve the minimum-length path routing problem in a unidirectional configuration.

Several important contributions presented in this work are as follows:

1. we have developed a new graphical model, called an access graph, for supporting the construction and/or search for paths in a unidirectional configuration.
2. the methods presented in this study not only find a path if one exists, but they also make track assignments such that the routability of potential future nets is enhanced. In this regard, our techniques encompass the primary attributes of cellular routes (e.g., Lee routes).
3. methods presented in this study reassign segments to tracks in order to make room for new nets. The graphical technique employed here supports this unique capability particularly well.
4. the methods presented in this study are useful in updating a routed board as well as routing a new configuration.

The fixed-track routing methods presented in Chapter 4 and analyzed in Chapter 9 are particularly useful in updating printed-circuit boards of the type considered in this work. Since the routed portion of the board is not disturbed during the routing process, new paths may be routed with minimum impact.

The floating-track routing methods presented in Chapter 5 are also useful in updating routed boards where existing routes are assigned new tracks during the routing of new paths. One of the methods presented (minimum-perturbation) minimizes the number of routed segments moved every time a new path is routed.

10.1 Suggested Research

1. Extend the research reported here to consider multiple-goal routing. For instance, the techniques presented in Chapter 4 should be adequate to permit the simultaneous minimization of wire length and congestion in $O(n^2)$ time. A possible initial approach could be to minimize both length and congestion in the minimum-path algorithm using the expression

$$af_1(\text{length}) + bg_2(\text{density})$$

where a and b are weights and f_1 and f_2 are functions of length and congestion respectively.

2. No attempt was made to examine the context in which unidirectional routing is performed in routing the entire board. The best method for obtaining the unidirectional net lists should be examined. Attempts at global optimization could be made.

3. Examine the performance of unidirectional routing to a greater depth than that done in Chapter 9. Some specific areas of examination are: fixed-track and floating-track performance both for new and changed configurations, route real problems, try various sortings on the net list before routing, and compare the results from the techniques given in Chapters 3, 4, and 5 with those given in Chapters 6, 7, and 8.

4. An assumption was made for fixed-track track assignment that minimizing the probabiltiy of future net conflicts by examining all possible two-point paths that could traverse the inner and outer channels of a segment to be assigned to a track would improve the completion rate of fixed-track routing. Investigate and analyze the validity of this assumption. Develop other ways to minimize the effect of routed path blockage as nets are routed.

BIBLIOGRAPHY

1. Breuer, M., "General Survey of the Design Automation of Digital Computers," Proceedings of the IEEE, vol. 54, pp. 1708-1721, December 1966.

2. Breuer, M., "Recent Developments in the Automated Design and Analysis of Digital Systems," Proceedings of the IEEE, vol. 60, no. 1, pp. 12-27, January 1972.

3. Kodres, U., "Partitioning and Card Selection," Chapter 4 in Design Automation of Digital Systems, Vol. 1 (ed. M.A. Breuer), Prentice-Hall, Englewood Cliffs, N.J., 1972.

4. Hanan, M., J. Kurtzberg, "Force-Vector Placement Techniques," IBM Report RC 2843, April 1970.

5. Steinberg, L., "The Backboard Wiring Problem: A Placement Algorithm," SIAM Review, vol. 3, no. 1, pp. 37-50, January 1961.

6. Lee, C., "An Algorithm for Path Connections and Its Applications," IRE Trans. on Elect. Computers, vol. EC-10, pp. 346-365, September 1961.

7. Moore, E., "Shortest Path through a Maze," Annals of the Computation Laboratory of Harvard University, vol. 30, pp. 285-292, 1959.

8. Hightower, D. "The Interconnection Problem — A Tutorial," Proceedings of the 10th Design Automation Workshop, pp. 1-12, 1973.

9. Akers, S., "Routing," Chapter 6 in Design Automation of Digital Systems, Vol. 1 (ed. M.A. Breuer), Prentice-Hall, Englewood Cliffs, N.J., 1972.

10. Mikami, K., K. Tabushi, "A Computer Program for Optimal Routing of Printed Circuit Connectors," IFIPS Proceedings, pp. 1475-1468, 1968.

11. Akers, S., "A Modification of Lee's Path Connection Algorithm," IEEE Trans. on Elect. Computers, vol. EC-16, pp. 97-98, February 1967.

12. Geyer, J., "Connection Routing Algorithm for Printed Circuit Boards," IEEE Trans. on Circuit Theory, vol. CT-18, no. 1, pp. 95-100, January 1971.

13. Vancleemput, W., J. Linders, "An Improved Graph-Theoretic Model for the Circuit Layout Problem," Proceedings of the 11th Design Automation Workshop, pp. 82-90, 1974.

14. Kuh, E.S., T. Kashiwahara, T. Fujisawa, "On Optimum Single-Row Routing," Electronics Research Memorandum No. UCB/ERL M78/26, University of California, Berkeley, May 22, 1978.

15. Foster, J.C., "A Router for Multilayer Printed Wiring Backplanes," Proceedings of the 10th Design Automation Workshop, pp. 44-49, 1973.

16. Stevens, J., "Fast Heuristic Techniques for Placing and Wiring Printed Circuit Boards," Ph.D. Thesis, University of Illinois, 1972.

17. Mah, L., L. Steinberg, "Topologic Class Routing for Printed Circuit Boards," Proceedings of the 9th Design Automation Workshop, pp. 80-93, 1972.

18. Lass, S., "Automated Printed Circuit Routing with a Stepping Aperture," Comm. ACM, vol. 12, no. 5, pp.

19. Hightower, D., "A Solution to Line Routing Problems in the Continuous Plane," Proceedings of the 6th Design Automation Workshop, pp.

20. Mathison, R., "A High Quality, Low Cost Router for MOS/LSI," Proceedings of the Design Automation Workshop, pp. 94-103, 1972.

21. Hitchcock, R., "Cellular Wiring and the Cellular Modeling Technique," Proceedings of the Design Automation Workshop, pp. 94-103, 1972.

22. So, H., "Some Theoretical Resutls on the Routing of Multilayer Printed-Wiring Boards," Proceedings of the IEEE International Symposium on Circuits and Systems, pp. 296-303, 1974.

23. Ting, B., E. Kuh, I. Shirakawa, "The Multilayer Routing Problem: Algorithms and Necessary and Sufficient Conditions for the Single-Row Single-Layer Case,"

IEEE Transactions on Circuits and Systems, vol. CAS-23, no. 12, pp. 768-778, December 1976.

24. Dijkstra, E.W., "A Note on Two Problems in Connexion with Graphs," Numerische Mathematik, vol. 1, pp. 269-271, 1959.

25. Rubin, F., "Printed Wire Routing for Multilayer Circuit Boards," Ph.D. Thesis, Syracuse University, June 1972.

26. Bondy, J.A., U.S.R. Murty, Graph Theory with Applications, American Elsevier Publishing Company, Inc., New York, 1976.

27. Harary, F., Graph Theory, Addison-Wesley Publishing Company, Reading, Massachusetts, 1972.

28. Aho, A., J. Hopcroft, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, Reading, Massachusetts, 1974.

29. Agrawal, P., "Routing of Printed Circuit Cards: Density, Analysis and Routing Algoirthms," Ph.D. Dissertation, University of Southern California, June 1977.

APPENDIX

COMMONLY USED SYMBOLS

a_i	an original point in a configuration
q_i	a pseudo-point
p_i	a unidirectional point (either an original point or a pseudo-point)
u_i	(1) an interval between two unidirectional points (p_i, p_{i+1}) (2) a node in a graph (used by Dijkstra's algorithm)
α_i	the location of point p_i
A	a set of original points $\{a_1, a_2, \ldots, a_m\}$
Q	a set of pseudo-points $\{q_1, q_2, \ldots, q_r\}$
P	a path in a graph
D	a unidirectional configuration
S^+	the upper street in a unidirectional configuration
S^-	the lower street in a unidirectional configuration
N	a net which contains exactly two end points
L	an ordered list of nets
G	an access graph
T^+	track capacity in the upper street S^+
T^-	track capacity in the lower street S^-
T	the number of free tracks in a channel

V	the set of nodes in a graph
E	the set of arcs (i.e., edges) in a graph
v_i	a node in the node set V of a graph
x	takes on the symbol "+" or "-" when used as a superscript
B	a set of values reflecting characteristics of a channel
w	represents a switching path, arc, or interval
δ	the length of a section channel
η	the number of nodes in a graph
e	the number of edges in a graph
S_i	a set of nodes used in Dijkstra's min path algorithm
$\ell(u_i)$	the total cost (i.e., weight) of a minimum-cost path from node u_0 to u_i
$w(u,v)$	the cost (i.e., weight) of an arc (u,v)
$d(u)$	the degree of node u
S	a stack
G_p	a precedence graph
G_s	a simplified access graph
s	a source node, interval, or point
t	(1) a terminal node, interval, or point (2) the track within a channel assigned to a feasible segment
G'_p	a reduced precedence graph
k	the number of arcs in an access graph
c	the number of segments in a configuration
m	(1) the number of original points a_i in a configuration

	(2) the number of nodes in a simplified access graph
z_i	a node in a path P
γ	a section
ρ	the number of empty tracks in a channel (i.e., channel capacity)
t_i	a track
μ_j	the inner tracks adjacent to a section whose left end point is p_j
D^*	a new routed configuration derived from a configuration D and a net not in D but routed in D^*
G'	a feasible access graph
g	(1) the number of switches in a feasible path in a configuration
	(2) a pointer used in Algorithm 4.2
$\hat{p}_i$	a feasible pseudo-point
$\hat{u}_i$	a feasible interval
z^x	a pointer used in Algorithm 4.2
Δ Σ	temporary symbols used in the construction of a feasible access graph
C	a set of unidirectional points $\{p_1, p_2, \ldots, p_n\}$
$\mathcal{P}$	the probability that a future two-point net will traverse a given channel
τ	a street track assigned to a feasible segment
Ω	the number of equally-likely future 2-point nets that can traverse a channel
π	the track occupied by the innermost segment of a channel
ψ	a channel
$\bar{G}$	a connected subgraph

H	(1) a biconnected subgraph of a connected graph (2) a conflict graph
β_i	(1) a node in a path (2) a temporary variable used in Algorithm 5.1
σ_i	the label of arc β_i
G^*	the access graph for a new routed configuration D^*
$d^+(p_i)$	the street density in street S^+ at point p_i
$d^W(a)$	the number of pseudo-points in interval a where a is bounded by original points
$\rho^+(u,v)$	the density of segment (u,v) in street S^+
λ	the number of routed segments that must be moved to accommodate a feasible c-section
ϕ	the number of pseudo-points in an interval a where a is bounded by original points
d	the depth of a stack
b	the number of two-point nets routed in a unidirectional configuration
B	the number of Lee cells in a unidirectional configuration