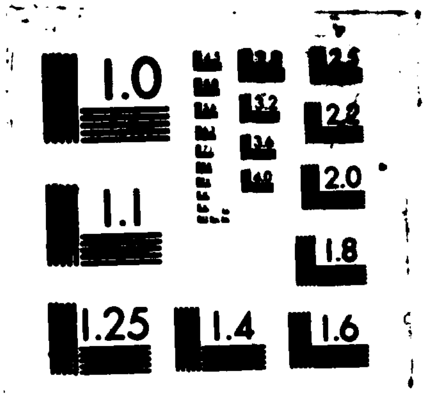
AD-A183 900 SOLVING A CLASS OF SPATIAL REASONING PROBLEMS: 1/5
MINIMAL-COST PATH PLANNING IN THE CARTESIAN PLANE(U)
NAVAL POSTGRADUATE SCHOOL MONTEREY CA R F RICHBOURG
UNCLASSIFIED JUN 87 F/G 12/9 NL


1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

SOLVING A CLASS OF
SPATIAL REASONING PROBLEMS:
MINIMAL-COST PATH PLANNING
IN THE CARTESIAN PLANE

by

Robert F. Richbourg

June 1987

Thesis Advisor: Neil C. Rowe

Approved for public release; distribution is unlimited.

unclassified

SECURITY CLASSIFICATION OF THIS PAGE

A183 900

REPORT DOCUMENTATION PAGE

1a. ORT SECURITY CLASSIFICATION classified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL *(If applicable)* 52	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000	7b. ADDRESS *(City, State, and ZIP Code)* Monterey, California 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL *(If applicable)*	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)*	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO | PROJECT NO | TASK NO | WORK UNIT ACCESSION NO

11. TITLE *(Include Security Classification)* SOLVING A CLASS OF SPATIAL REASONING PROBLEMS: MINIMAL-COST PATH PLANNING IN THE CARTESIAN PLANE

12. ONAL AUTHOR(S) Richbourg, Robert F.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Doctoral Thesis	FROM ________ TO ________	1987 June	429

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Spatial reasoning; Snell's Law; path planning; artificial intelligence; search; geodesic

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This work presents an algorithm to solve a two-dimensional weighted-region problem that requires finding the least-cost path between two points located on a map of homogeneous-cost regions. Such regions have a constant cost rate per unit distance accrued by paths passing through them. Conventional graph search applies standard search strategies to graphs whose links represent the only possible paths. We use Snell's law as a local-optimality criterion to create corresponding graphs for the weighted-region problem; the nodes in our graphs represent areal subdivisions of the physical environment. The performance of our Snell's-law-based algorithm is compared to that of a dynamic-programming, wavefront-propagation technique. Test results show average-case superiority of the Snell's-law-based algorithm, as measured by time, space and solution-path cost. We present a criterion to predict the time for the wavefront-propagation

20. IBUTION/AVAILABILITY OF ABSTRACT CLASSIFIED/UNLIMITED □ SAME AS RPT □ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Prof. Neil C. Rowe	22b. TELEPHONE *(Include Area Code)* (408) 646-2462 22c. OFFICE SYMBOL Code 52Rp

83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE
unclassified

BLOCK 19 CONTINUATION

algorithm and the Snell's-law algorithm to solve problems; this allows the selection of the fastest algorithm. We also develop improvements to the wavefront-propagation algorithm that decrease its average-case time requirements and we prove properties of Snell's law when applied to the weighted-region problem.

Approved for public release; distribution is unlimited

Solving a Class of Spatial Reasoning Problems: Minimal-Cost Path Planning in the Cartesian Plane

by

Robert F. Richbourg
Major, United States Army
B.S., Wake Forest University, 1976
M.S., Naval Postgraduate School, 1984

Submitted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1987

Author: Robert F. Richbourg

Approved by:

Uno R. Kodres

Uno R. Kodres
Professor of Computer Science

Robert B. McGhee
Professor of Computer Science

Michael J. Zyda
Associate Professor of Computer Science

Douglas R. Smith
Computer Scientist, Kestrel Institute

Gilbert T. Howard

Gilbert T. Howard
Associate Professor of
Operations Research

Neil C. Rowe

Neil C. Rowe
Associate Professor of Computer Science
Dissertation Supervisor

Approved by:

Vincent Y. Lum, Chairman, Department of Computer Science

Approved by:

David A. Schrady, Academic Dean

ABSTRACT

This work presents an algorithm to solve a two-dimensional weighted-region problem that requires finding the least-cost path between two points located on a map of homogeneous-cost regions. Such regions have a constant cost rate per unit distance accrued by paths passing through them. Conventional graph search applies standard search strategies to graphs whose links represent the only possible paths. We use Snell's law as a local-optimality criterion to create corresponding graphs for the weighted-region problem; the nodes in our graphs represent areal subdivisions of the physical environment. The performance of our Snell's-law-based algorithm is compared to that of a dynamic-programming, wavefront-propagation technique. Test results show average-case superiority of the Snell's-law-based algorithm, as measured by time, space and solution-path cost. We present a criterion to predict the time for the wavefront-propagation algorithm and the Snell's-law algorithm to solve problems; this allows the selection of the fastest algorithm. We also develop improvements to the wavefront-propagation algorithm that decrease its average-case time requirements and we prove properties of Snell's law when applied to the weighted-region problem.

TABLE OF CONTENTS

ACKNOWLEDGMENT

Primarily, I wish to thank my wife Alice for her unfailing support. Without her help, the completion of this work would have required much more time. I would also like to thank Dr. Neil Rowe for his attention to detail and for his insightful guidance. The intellectual maturity of Dr. Robert McGhee and enthusiastic support of Dr. Michael Zyda were also greatly appreciated. Finally, I wish to recognize Dr. David Hsiao for his initiative in establishing a computer science doctoral program at the Naval Postgraduate School.

This work was supported in part by the U.S. Army Combat Developments Experimentation Center (USACDEC) under MIPR ATEC 88-86.

I. INTRODUCTION

A. SPATIAL REASONING

Robotics has been characterized as the field concerned with the "intelligent connection of perception to action". [Ref. 1]. A key to establishing the connection between computers and human-like activity involves *spatial reasoning;* reasoning about objects based on descriptions of their spatial properties such as location and shape [Ref. 2]. As an example, suppose that a robot manipulator is used to take parts from one bin and put the parts into another bin. Before this task can be accomplished, the robot must "know" the location, size and shape of both bins as well as how to move its arm between bins.

Spatial reasoning problems can be varied in nature; there does not seem to be a single spatial reasoning problem that represents the entire set of such problems. One class of spatial reasoning problems that has received much attention is path planning. That is, given a map of the physical environment that provides the location, shape and size of distinct regions and associates a cost per unit distance with moving through each such region, find the least-cost path between two given points on the map. Clearly, if a robot is to move its arm (between bins for example), it must be able to solve an instance of the path-planning problem.

Optimal-cost route planning is not a new problem area. There have been many successful *search* techniques developed, particularly in the operations research field, to solve different instances of path-planning problems. Search is the process of exploring different alternatives that lead to or constitute solutions. Normally, the techniques tacitly rely on a strong assumption: a finite graph that exhaustively lists every possible path in the environment is either available or can be generated. This implies that there is a finite number of turns that can be taken at every branch point (i.e., each node in the graph has a finite branching factor). This assumption is reasonable for many path-planning domains. As an example, when the problem is to plan a roadway route between two locations, then a graph that represents the road network connecting the two locations is an accurate problem representation. However, a finite graph where graph nodes represent locations and graph links define avenues for movement between locations, does not exist for all path-planning problems.

The type of graph structure discussed above is an example of a problem representation that facilitates a *path-oriented* approach [Ref. 2] to the path-planning problem. To illustrate the inadequacies of the path-oriented approach, suppose that an optimal-cost route between two locations is desired and that the locations are connected by a road network. If the agent (i.e., some entity capable of independant motion) for which the path is to be planned is not restricted to road-only travel, then the road network does not exhaustively represent all possible paths between the two locations. If the agent happens to be a human on foot, a roadway may not be a desirable terrain feature to include on the path. For example, if the human wishes to avoid detection, a wooded area would be preferable to an open road.

In general, a solution path appropriate for a highly-mobile agent includes path segments that cross several different terrain features. Again, using the example of a walking human, the most desirable paths often combine some roadway and off-road portions. Such solution paths do not seem to come from selecting one from a finite number of possible paths through a graph. If a road is considered to be a series of connected line segments, then due to the existence of real numbers, there exists an uncountably infinite number of points where a path could exit the road to begin an off-road path segment. Thus, there exists an uncountably infinite number of possible paths involving on-road and off-road path segments. Clearly, a finite graph where one node has an infinite number of neighboring nodes is self-contradicting.

But, even though there exists an infinite number of possible paths comprised of on-road and off-road path segments, the differences between all but a finite number of them cannot be represented on any machine that has finite precision. So, a finite static graph that reasonably closely models all possible paths has to be large. The alternative is to decrease the resolution (precision) of the representation, resulting in solutions of decreased accuracy. In many domains, the type of solution that can be achieved based on simplified approximate problem representations is satisfactory. However, in some instances of the path-planning problem, where path cost is measured in terms of exposure to danger for example, sacrificing optimality for computational simplicity is not a good strategy.

Another more promising method of solving path-planning problems in domains where the path-oriented approach is inadequate involves *shape-oriented* reasoning [Ref 2]. Instead of relying on the search of a large graph that includes links for all possible paths between two locations, reason about the spatial relations and properties of the terrain features themselves (as represented by regions having uniform properties). Shape-oriented spatial reasoning can be used to create a graph that represents areal combinations of different terrain features. In this work, we present methods of creating and searching such spatially-oriented graphs that allow solutions to the path-planning problem.

B. PROBLEM DEFINITION AND BASIC ASSUMPTIONS

The path-planning problem that we solve has been named the *weighted-region problem* [Ref. 3] and requires finding the least-cost path between two given points, a *start* and a *goal* that both lie in the same Cartesian plane. We assume, as a given, the existence of a *area-cost map* that is large enough to include the start, the goal, and the least-cost path between them. The area-cost map is comprised of *homogeneous-cost regions,* described as non-intersecting polygons such that each polygon defines an area of equal *cost rate.* A cost rate is a generic measure of cost per unit distance, generic in the sense that the unit of measure itself is irrelevant and could be, for example, time, exposure to danger, energy required, or a similar unit of measure. Cost rates are defined only in terms of location (i.e., not in terms of heading or time) and for a specific *agent.* By agent, we indicate some entity capable of independent motion over the area represented by the area-cost map. There is a single cost rate associated with each homogeneous-cost region.

A *path* is a series of connected line segments or *path segments,* that begins at the start and ends at the goal. There is one path segment on a path for each portion of the path that is inside a single homogeneous-cost region. Thus, a path is comprised of path segments such that there is one path segment endpoint at the start, one path segment endpoint at the goal, and one path segment endpoint at each point where the path intersects a boundary of a homogeneous-cost region. The cost of a path can be calculated by summing the costs of each of its path segments. Since each path segment goes through a single homogeneous-cost region, a single cost rate can be associated with each path segment. The cost of a path segment is equal to the cost rate for that segment multiplied by the length of the segment.

Let P_{SG} be the set of all simple, start-to-goal paths for a specific instance of a least-cost-path problem. For each $p_i \epsilon P_{SG}$, let p_i be the locus of points (x,y) such that $x = h_i(s)$ and $y = g_i(s)$. Let $C(x,y)$ be a piecewise constant function such that $TC = C(x,y)$ is a unique cost rate (per unit distance) associated with coordinates (x,y) on the area-cost map. Then, the least-cost-path problem can be expressed as:

$$\min_{p_i \epsilon P_{SG}} \int_{p_i} C(h_i(s),g_i(s))\ ds$$

Note that this formulation represents a problem in the calculus of variations [Ref. 4]. In this work, we exploit the nature of the problem itself to devise a solution technique that is based on less complex mathematical models. Specifically, we define a local optimality criterion that allows the computation of piecewise-linear paths from the start to the goal that have locally optimal cost, locally optimal among the set of all start-to-goal paths that intersect a particular set of homogeneous-cost region boundaries. The problem then reduces to finding the single least-cost path from the set of all locally optimal-cost paths. Let L be the set of all such locally optimal-cost paths such that $l_i \epsilon L$ and that each l_i is comprised of n_i path segments. denoted ps_j (where $j \epsilon \{1,...,n_i\}$). A unique cost.

c_j can be associated with each such path segment. Let the Euclidean distance along path segment ps_j be denoted d_j. Given this formulation, the weighted-region problem becomes:

$$\min_{l_i \epsilon L} \sum_{j=1}^{n_i} c_j d_j$$

C. THESIS ORGANIZATION

In the following chapter, we examine solution techniques for some related problems, beginning with general graph search strategies. The chapter includes a brief discussion of the wavefront-propagation technique as a solution method for the weighted-region problem. Because wavefront propagation is a widely used method, Chapter III is devoted to a detailed discussion of the method as well as some modifications that can enhance its performance.

In Chapter II, the survey of related work also includes a discussion of a weighted-region problem solution technique that relies on Snell's law, commonly used in optics. Chapter IV presents a mathematical analysis of the application of Snell's law to the weighted-region problem. The properties exhibited by Snell's law when applied to the weighted-region problem that are developed in Chapter IV provide the foundation for the Snell's-law-based algorithm developed in Chapter V. A prototype version of the algorithm has been implemented. In Chapter VI, we present performance comparisons of the Snell's-law-based algorithm and the wavefront-propagation technique. This chapter provides the data that is the basis of the conclusions presented in Chapter VII.

II. SURVEY OF PREVIOUS WORK

A. INTRODUCTION

Recall that the weighted-region problem requires finding the optimal-cost path between two known points given an appropriate area-cost map. The weighted-region problem is related to several other problems in the fields of computer science and operations research. Solving the weighted-region problem requires search. There are many search strategies that can be applied to the problem. Each strategy has unique characteristics that determine its suitability. In Section II.B, we discuss the characteristics of well-known search strategies that have been applied to the weighted-region problem.

Work completed in the artificial intelligence field has demonstrated that determining the most suitable search strategy for a particular problem is but a single step towards constructing a problem solving system. Many other aspects are involved. Problem representation, discussed in Section II.C, is one important aspect. In Section II.D we discuss these issues in the context of problem solving systems created by artificial intelligence researchers.

The discussions presented in Sections II.B and II.D develop basic principles for constructing problem solving systems that rely on search. Section II.E exemplifies the application of these principles to the simplest instance of the weighted-region problem. This restricted version of the problem requires finding optimal-cost paths through an environment consisting only of obstacle areas (those areas having infinite costs associated with passing through them) and traversable areas (those areas having finite constant costs associated with passing through them).

Section II.F presents completed work that has been directed towards solving the general-case weighted-region problem. Each technique has its own characteristic advantages and disadvantages. These characteristics are developed leading up to Section II.G, the summary.

B. WEAK METHODS OF SEARCH

Search is required whenever there is no closed-form solution for the problem at hand. That is, if the best that can be done when presented with a problem is to make a plausible guess at its solution, search is required to solve the problem. An oversimplification of this statement is that search is required whenever a complex problem having many plausible alternative solutions, must be optimally solved. Thus, search is a fundamental requirement in producing solutions for many important problems. Because search problems are ubiquitous, many different search strategies have been developed and are well understood. These strategies are often called *weak methods* [Ref. 5]. The term weak method is not meant to reflect problem solving power. Rather, the classification indicates that the method does not have a strong reliance on a particular aspect of a problem, the *problem structure* and is thus generally applicable to a wide range of problems.

At least three requirements must be met by every problem-solving system before a weak method can be successfully applied. First, there must be some way of describing the problem and its subparts. That is, there must be a *problem representation* that describes every object or *state* that is involved in solving the problem. Secondly, there must be some way of specifying motion, or transformations, from one state to another. *Operators* are state modification rules that specify how a state can be transformed and describe states resulting from transformations. Finally, there must be some method of ordering operator application. A *control strategy* establishes a precedence among the operators.

Constructing efficient operators and problem representations are important issues that are discussed in Section II.C. In this section, we focus on control strategies. To facilitate the discussion, we assume that the problem representation is a graph consisting of nodes and links. Each node represents a state. There is a link in the graph between each two states when they are related by a single application of an operator. Using this graph structure allows the explanation of basic terminology.

Node A is a *parent* of node B if there is a directed link from A to B. (Thus, the state represented by A can be transformed into the state represented by B by a single application of an operator.) In this case, node B is a *child* of node A. If there is a chain of links leading from node A to node B then node A is an *ancestor* of node B and node B is a *successor* of node A. Two or more child nodes that have the same parent node are *siblings*. A fundamental step in graph search is node *generation*. A node is generated when it has been derived by traversing the link from its parent node. Once a node has been generated, the parent of that node has been *explored*. *Expanding* a node requires generating all of the node's children. The minimum number of links from a node back to the start is the *depth* of the node.

Weak methods can be characterized by the type of solutions they produce. The solution type is normally dictated by the task at hand. The task can be an optimizing, satisficing or semi-optimizing task [Ref. 6]. If the task requires finding the exact most desirable solution, then it is an optimizing task. The task is satisficing when solutions that are "good enough" [Ref. 7] are acceptable. Satisficing tasks usually involve the use of *heuristics,* or rules of thumb which serve to lessen the search effort. Here, optimal solutions are not guaranteed. Often, the first solution found during search is acceptable for a satisficing task. When the task is to find a solution that is within a specified tolerance of the exact optimal solution, the task is semi-optimizing. The latter type of task appears frequently for several reasons. First, a task can be semi-optimizing because the effort required to improve a solution that is close to optimal is not justified by the amount of improvement. Secondly, a task can be semi-optimizing due to numerical issues such as precision and accuracy. If the exact optimal solution is, for example, π, no machine currently available can exactly display the number. The weighted-region problem is a semi-optimizing task for both of these reasons. As such, a search strategy suitable for the weighted-region problem must support semi-optimizing tasks. Normally, this implies that the strategy be capable of providing optimal solutions, given an unlimited amount of resources.

Weak methods can also be characterized by the nature of their control strategy. A control strategy is *systematic* if it is both *complete* and *non-redundant* [Ref. 6,8]. Completeness implies that a solution, if one exists, will not be overlooked. Non-redundancy implies that the search will not repeat itself by exploring any alternative more than once. Control strategies that do not meet these requirements are *non-systematic*. Clearly, a control strategy suitable for the weighted-region problem must be systematic. Since the task is semi-optimizing, the strategy must be complete. Non-redundancy ensures only the most rudimentary form of efficiency.

Thus, we desire a weak method that is systematic and appropriate for a semi-optimizing task. Several weak methods have been applied to the weighted-region problem. Prior to discussing these methods, we discuss the primitive search strategies that form the basis for the more advanced weak methods of search. (Note that, unless otherwise stated, our discussions of the following search strategies assume that the first solution a strategy finds is the solution that the strategy returns.)

The two simplest systematic strategies for conducting a graph search are *depth-first search* and *breadth-first search* [Ref. 6]. To illustrate the differences in these strategies, suppose that node N has two child nodes, C1 and C2. Assume that node N has been explored. When using a depth-first strategy, if C1 is selected for expansion before C2, then all of the successors of C1 will be explored before C2. Thus, depth-first search choses for expansion first those nodes that increase search depth in the graph. Breadth-first search uses an opposite philosophy. All nodes at the same depth in the graph are expanded before the search moves to a greater depth in the graph. Thus, all siblings of node N would be explored before either C1 or C1 in the above example.

When depth-first search reaches a node that has no children, it must *backtrack* by exploring sibling nodes. Thus, if node C1 is selected for expansion but C1 has no children, a depth-first strategy would explore node C2 next (in a depth-first manner). Since graphs do not have infinite branching factors, there is no breadth-first analogy to backtracking. However, both strategies are subject to *cycling*. When there is more than one path to a node (including the case where a node is a successor of itself) either strategy can cycle, or repeat the search effort by generating the same node more than one time. To prevent such occurrences, both strategies rely on maintaining two sets of nodes called *Open* and *Closed* [Ref. 8]. Open is the set of all nodes that have been generated but not yet expanded. Closed is the set of all nodes that have been expanded. Cycling is prevented by inspecting each node selected for expansion to ensure that it is not already in the Closed set. Generally, when the node is a member of Closed, it need not be expanded a second time and can simply be removed from the Open set.

Whenever either strategy returns the first solution that they find, they act as satisficing techniques. Heuristics can be added to either strategy to ensure that optimal solutions are discovered. Optimality can also be ensured by non-heuristic means. Either strategy can be used to conduct an *exhaustive* search of the graph so that the entire graph is searched. In an exhaustive search, all solutions will be found and the single best solution can be returned as optimal.

1. Uniform-Cost Search

Uniform-cost search is a type of breadth-first search that is suitable for optimizing tasks when different costs are associated with traversing links. Recall that breadth-first search exhaustively explores a graph at one level of depth before progressing to the next level. Uniform-cost search employs the same strategy except that the graph is explored in equal levels of cost rather than equal levels of depth. To effect the change, only a small modification is required to a procedural definition of breadth-first search. Instead of directly adding all children of a newly expanded node to the Open list, the children are first sorted into a new list, ordered by increasing cost to reach the child from the start node. The new list is then merge-sorted into the Open list. Cost information must be available for each node on the Open list to complete the merge sort. Thus, it is most convenient to require the elements of Open to be two-tuples of the form (Cost,Node).

The merge sort maintains Open so that the least-cost element is always the first element. Thus, the graph is searched by always expanding the least expensive path found so far. Clearly, when the goal node is the first element on Open, it represents the least expensive goal path that has yet been found and it can be returned as the optimal-cost solution. A procedural definition of uniform-cost search is presented in Table 1.

Uniform-cost search has also been called a *branch-and-bound* strategy since the name is descriptive of the strategy's behavior. The node chosen for expansion always represents a lower bound on the cost to reach the goal. If that node is not the goal, the strategy branches to some other, possibly unrelated, node on the next expansion. The branch-and-bound name is often used for this strategy in the artificial intelligence community. Branch-and-bound is also an archetypal search strategy used in operations research. However, the operations research version of branch-and-bound is a different algorithm, similar to the A* strategy (discussed in Section II.B.3 below). Uniform-cost search has also been called Dijkstra's algorithm after E. Dijkstra who first developed the strategy in 1956 [Ref. 9].

TABLE 1
UNIFORM-COST SEARCH

```
Uniform_cost(Closed,Open)
   {
   If Open is empty, stop, announcing no solution exists
   Otherwise
      {
      Split Open into 2 parts, Node which is the first element
         of Open and RestOpen which is Open with the first
         element removed.
      If Node is the goal and all other nodes having lower cost
         have been expanded, stop, announcing success
      Otherwise
         {
         Expand Node
         Create UpdatedClosed by adding Node to Closed
         If Node has no successors
            {
            Uniform_cost(UpdatedClosed.RestOpen)
            }
         Otherwise
            {
            Create the list SortedChildren which contains each
                child of Node, not already on Closed or Open,
                sorted by path cost from the start.
            Create UpdatedOpen by merging RestOpen and
                SortedChildren.
            Uniform_cost(UpdatedClosed,UpdatedOpen)
            }
         }
      }
   }
```

Uniform-cost search is a strategy that rests on the dynamic programming paradigm. Clearly, the first path found to any node constitutes the optimal path to that node. Thus, in the process of finding the optimal start-to-goal path, the optimal path to every node (when the secondary paths have cost less than the solution cost) expanded along the way is also found. This is analogous to the dynamic programming principle of solving all subproblems in order to solve the overall problem.

Uniform-cost search is classified as an *uninformed* (or "blind") strategy. (In fact, when the cost to reach any child node from its parent node is constant, uniform-cost search reduces to breadth-first search.) In the context of a graph-structured problem representation, uninformed strategies are those that have no notion of the location of the goal until it is found. That is, in uninformed strategies, the order of node expansion is not affected by the location of the goal node. Clearly, depth-first, breadth-first and exhaustive search strategies are also uninformed. Efficiency considerations normally imply that uninformed strategies are impractical for application to problems represented by large graph structures. However, the methods are important. They form the basis for more sophisticated, *informed* control strategies. The weak methods presented in the following three sections are all informed strategies.

2. Best-First Search

Best-first search is the basic informed strategy. The technique relies on heuristics that evaluate newly generated nodes in terms of their estimated proximity to the goal node. That is, if nodes A and B are newly generated and node A seems closer to the goal than B, the heuristic component of the algorithm will rate A as more promising than B and A will be expanded before B.

Best-first search provides a way to combine breadth-first and depth-first search. Procedurally, best-first search is a variant of the uniform-cost strategy. Exactly the same algorithm may be used except that a modification must be made to the method of evaluating the cost associated with each node. Instead of maintaining a "running" account of the cost to reach a node, a heuristic component that evaluates the "closeness" of the node to the goal is required. In the simple version of best-first search that we define here, the accrued cost to reach a node is unimportant. (We note that more sophisticated versions of best-first search rely on a "composite" evaluation at each node that includes a "running" account component and a "closeness" component. A* search, discussed below, is such a best-first strategy.) Once the cost evaluation for each node has been established, the nodes are again ordered in terms of increasing cost and merged into the Open list. Thus, the node that seems closest to the goal is always selected as the next node to expand. Again, the merge-sorting requirement dictates that elements of Open are (Cost,Node)-tuples. A procedural definition of best-first search is provided in Table 2.

The accuracy of the heuristic evaluation function has a great effect on the efficiency of best-first search. If the estimator is perfect, always returning the exact cost to reach the goal from any node, then there is no search involved in the problem. When using a perfect estimator, one child of each node expanded must be on the solution path and that child must be closer to the goal than the parent. Thus, no node not on the optimal path is ever expanded. Perfect estimators allow best-first search to behave as if it were depth-first search that always happens to choose the correct node for expansion. However, problems that require search (i.e., do not have closed form solutions) do not have perfect estimators. Normally, best-first search alternates between breadth-first and depth-first exploration of a graph.

TABLE 2
BEST-FIRST SEARCH

```
Best_first(Closed.Open)
  {
  If Open is empty, stop, announcing no solution exists
  Otherwise
    {
    Split Open into 2 parts, Node which is the first element
      of Open and RestOpen which is Open with the first
      element removed.
    If Node is the goal, stop, announcing success
    Otherwise
      {
      Expand Node
      Create UpdatedClosed by adding Node to Closed
      If Node has no successors
        {
        Best_first(UpdatedClosed.RestOpen)
        }
      Otherwise
        {
        Create the list SortedChildren which contains each
           child of Node, not already on Closed or Open,
           sorted in order of increasing estimates to reach
           the goal from the child.
        Create UpdatedOpen by merging RestOpen and
           SortedChildren.
        Best_first(UpdatedClosed,UpdatedOpen)
        }
      }
    }
  }
```

The heuristic estimator also determines the type of solution returned by our simple version of best-first search. If the estimator is perfect, optimal solutions are provided. Without perfect estimators, our best-first search is not guaranteed to provide optimal solutions. Overestimating the cost to reach the goal from a node on the optimal solution path will cause the expansion of that node to be delayed. Some other path leading to the goal can be found during the delay if the overestimate is sufficiently high. Similarly, a heuristic component that underestimates remaining cost can lead to less than optimal solutions. In this case, the problem stems from ignoring accrued cost to reach a node. As an example, suppose that node N is expanded, yielding children C_1 and C_2, both of which have the goal node, G, as a child. Let the cost of the $N-C_1$ link be 10 units while the cost of the $N-C_2$ link is 20 units. Let the cost of the C_1-G link be 15 units and the cost of the C_2-G link be 10 units. Assume that the heuristic component underestimates the remaining cost from both C_1 and C_2, but is accurate enough to prefer C_2 to C_1 since the former is closer to the goal. The path through C_2 will be explored first, yielding a solution having a cost of 30 units. The path through node C_1 is better, having a cost of 25 units, but that path is not explored.

Our simple best-first search is appropriate for satisficing tasks when the size of the problem representation requires an informed strategy and there is a good estimator available. The strategy is not suitable for optimizing or semi-optimizing tasks. However, our simple best-first search can be modified to provide optimal solutions. The modification is presented in the next section.

3. The A* Search Strategy

A* search [Ref. 10] is the culmination of all the strategies that have thus far been presented. The strategy combines the informed nature of best-first search with the optimizing character of uniform-cost search. As a result, whenever a good estimator is available, A* search is the leading candidate for the best strategy to apply to optimizing or semi-optimizing tasks.

The A* strategy relies on maintaining a composite cost evaluation for each node stored on the Open list. The composite is the sum of the known cost to reach the node (as in uniform-cost search) and a heuristic lower-bound underestimate of the cost to reach the goal from that node (similar to the evaluation used by our best-first search). The normal names given to these costs are f(N) for the composite cost at node N, g(N) for the known cost to reach node N and h(N) for the lower-bound heuristic evaluation component. Thus, f(N)=g(N)+h(N) in this terminology. A* search overcomes the inefficiency of uniform-cost search by using the lower-bound component and changes the satisficing nature of best first search to optimizing through the known-cost component.

The above is an oversimplification. A* search cannot provide an optimal solution unless the lower-bound component is guaranteed to underestimate remaining cost. That is, the lower-bound component must provide an evaluation that is less than or equal to the actual remaining cost to reach the goal. In general, if the lower-bound component overestimates remaining cost, a node on the optimal path can be overlooked (as in the case of best-first search). However, using a lower-bound component that underestimates remaining cost guarantees that A* returns optimal solutions.

The amount that the lower-bound function underestimates true remaining cost affects the work required by A* to reach a solution. The closer the evaluator comes to perfection, the less work required by A*. Altering the values returned by either the known-cost or lower-bound function changes the behavior of the A* algorithm. These behavioral changes are summarized in Table 3.

TABLE 3
ALTERING A* BEHAVIOR

known cost	lower bound	A* Behavior	Optimality Guaranteed
true cost	true cost	depth first, no backtracking	yes
true cost	underestimate	-	yes
true cost	overestimate	-	no
true cost	0	uniform-cost search	yes
0	0	random	no
0	any estimate	simple best-first search	no

A* search, like the other strategies presented, requires use of both the Closed and Open lists to prevent cycling. However, there is an important difference in how the lists are maintained when a duplicate node is generated (i.e., a node already on Closed). Unlike uniform-cost search, A* may not generate the optimal path to an intermediate node the first time it finds a path to that node. This is due to the underestimate provided by the lower-bound component. The evaluation is based on remaining cost to reach the goal node, not on remaining cost to any intermediate node. As an example, suppose that nodes N_1 and N_2 both have a single child, C, and that the known cost for N_1 equals the known cost for N_2. Let the actual cost from N_1 to C be 5 units while the cost from N_2 to C is only 3 units. If the lower-bound evaluation for N_1 is less than the lower-bound evaluation for N_2, then the path reaching C through N_1 will be generated prior to the less expensive path through N_2.

To overcome this problem, care must be taken when the same child node is generated by different parent nodes. If there is a newly generated node, N, such that the new known cost for N equals G1 and N is already on the Open list with an old known-cost value of G2, then comparing G1 and G2 determines the correct action. If $G2 \leqslant G1$, then the newly generated instance of N can be discarded. If the converse is true, the new instance of N must replace the old instance of N on Open so that the least-cost ancestry of N is recorded. If the replacement does not occur, the effect of the incorrect known-cost value for N is propagated to all successors of N, which could cause the optimal solution path to be overlooked. The situation is more complicated if the newly generated node already appears on the Closed list.

When a newly generated node, N, is a duplicate of an explored node stored on the Closed list, the known-cost functions for both occurrences of N must again be compared. Again, let the new instance of N be such that the new known-cost value for N is G1 while the old instance of N has a known-cost value of G2. If $G2 \leqslant G1$, then no actions are required and the new instance of N can simply be discarded. If $G1 < G2$, the ancestry records of the instance of N appearing on Closed must be updated. Moreover, N can be an ancestor of other nodes appearing on either Closed or Open. The cost difference for the known-cost value at the new instance of N must be propagated to each of the previously generated successors appearing on either list.

There are two options to update the known-cost values for the successor nodes. First, the procedure outlined above may be followed, finding each successor and updating its known-cost value and ancestry records. This option is advantageous if the cost of generating successor nodes is relatively high. The second option is to simply remove the instance of the duplicate N from Closed and insert the new instance of N on the Open list as normal. This option avoids tracing through the ancestry records of nodes on Open and Closed at the cost of regenerating some successor nodes. In many cases, the cost of expanding a node is low, particularly when a graph structured problem representation is used. Thus, our procedural definition of the A* strategy employs the latter option. (We note that "Closed" is not the best name for the list of expanded nodes since the nodes may be regenerated. However, we use this terminology for historical reasons.)

The requirement to compare known-cost values for duplicate generations of the same node means that these values must be available. Thus, each element of Open must now be a tuple of the form (f(N),g(N),N) which is a short notation for (composite evaluation, known-cost value, lower-bound evaluation). The f(N) (i.e., composite evaluation) values are required to sort the Open list. The Closed list must now be comprised of elements having the form (g(N),N). A procedural definition of A* is provided in Table 4.

4. Epsilon-Admissible A* Search

A* search as described in the preceding section is an appropriate strategy for optimizing tasks. However, there is often a significant overhead involved in computing and maintaining heuristic evaluation (i.e., the lower-bound h(N) values). A natural question arises: is there a way to improve performance of an A* strategy while accepting only a bounded decrease in solution quality? An algorithm founded on this premise is well suited to the many interesting problems that have semi-optimizing requirements.

Epsilon-admissible speedup A*, normally written $A_\epsilon{}^*$, is a semi-optimizing strategy that guarantees its solution has cost less than or equal to $(1 + \epsilon)$ times the cost of the exact optimal solution [Ref. 6]. The value of ϵ can be specified for

TABLE 4
A* SEARCH

```
A*(Closed,Open)
  {
  If Open is empty, stop, announcing no solution exists.
  Otherwise
    {
    Split Open into two parts,(f(Node),g(Node),Node) which
    is the first element of Open and RestOpen which is Open
    with the first element removed.
    If Node is the goal, stop, announcing success.
    Otherwise
      {
      Expand Node.
      Create UpdatedClosed by adding (g(Node),Node) to Closed.
      For each child, C, of Node
        {
        calculate g(C), h(C) and f(C).
        if C is on Closed with a g(N) value of G2 and
            $G2 \leqslant g(C)$, then discard C.
        otherwise, if C is on Closed, remove the instance
            of C from Closed and merge (f(C),g(C),C) into
            RestOpen.
        otherwise, if C is on RestOpen with a g(N) value of
            of G2 and $G2 \leqslant g(C)$, discard C.
        otherwise if C is on RestOpen, delete the old instance
            of C on RestOpen and merge (f(C),g(C),C) onto
            RestOpen.
        otherwise merge (f(C),g(C),C) into RestOpen.
        }
      A*(UpdatedClosed,RestOpen)
      }
    }
  }
```

any particular problem. A_ϵ^* operates in the same manner as A* except that two heuristic functions h(N) and h_focal(N) are required and one new list, normally called Focal, is maintained in addition to Closed and Open. Focal is a subset of Open. Let the first node on Open be N_{min}. This node, by definition, has the lowest composite evaluation (i.e., f(N) value) of all nodes on Open. N_{min} is a member of Focal. Every other node on Open that has a composite evaluation less than or equal to f(N_{min}) + ϵ is also a member of Focal.

Instead of selecting N_{min} directly from Open to be the next node expanded, all nodes on Focal are rated by a second heuristic function, h_focal(N). The member of Focal that receives the lowest h_focal(N) rating is chosen as the next expanded node. The heuristic h_focal(N) is intended to estimate the work required to arrive at a solution from node N. Often, h_focal(N) is identical to the lower-bound evaluation, h(N), reflecting the theory that the node closest to the goal should require the least amount of work to complete a solution path. When h(N) = h_focal(N), the procedure examines successive members of Open, beginning with N_{min}, until the composite evaluation of the member exceeds f(N_{min}) + ϵ. Of these nodes, the one with the lowest lower-bound evaluation is selected for expansion. If multiple nodes have the same lower-bound evaluation, the node with the lowest composite evaluation is selected. Table 5 provides a procedural definition of A_ϵ* search.

Other than the process used to select the next expanded node, A_ϵ* operates exactly as does the A* algorithm. In many cases, A_ϵ* is more efficient than A* since some nodes can be "skipped over" in the expansion process. Thus, A_ϵ* is often more appropriate for semi-optimizing tasks. This is particularly true when there are many feasible solutions whose cost is very close to optimal. Then, the A_ϵ* algorithm avoids the necessity of finding each such solution by returning the first solution found to be within ϵ of the optimal solution.

TABLE 5
EPSILON-ADMISSIBLE A* SEARCH

```
Epsilon_A*(Closed,Open)
   {
   If Open is empty, stop, announcing no solution exists.
   Otherwise
      {
      Let Focal be the empty list
      Set f_min to be the f(N) value of the first node on Open
      While Node on Open has $f(N) < f\_min + \epsilon$
         {
         Evaluate h_focal(Node)
         Merge Node onto Focal, in order of increasing
            h_focal(Node) values
         }
      Split Open into two parts,(f(Node),g(Node),Node) which
      is the first element of Focal and RestOpen which is Open
      with the first element of Focal removed.
      If Node is the goal, stop, announcing success.
      Otherwise
         {
         Expand Node.
         Create UpdatedClosed by adding (g(Node),Node) to Closed.
         For each child, C, of Node
            {
            calculate g(C), h(C) and f(C).
            if C is on Closed with a g(N) value of G2 and
                $G2 \leqslant g(C)$, then discard C.
            otherwise, if C is on Closed, remove the instance
                of C from Closed and merge (f(C),g(C),C) into
                RestOpen.
            otherwise, if C is on RestOpen with a g(N) value of
                of G2 and $G2 \leqslant g(C)$, discard C.
            otherwise if C is on RestOpen, delete the old instance
                of C on RestOpen and merge (f(C),g(C),C) onto
                RestOpen.
            otherwise merge (f(C),g(C),C) into RestOpen.
            }
         Epsilon_A*(UpdatedClosed,RestOpen)
         }
      }
   }
```

C. PROBLEM REPRESENTATION

Problem representation is a core area within artificial intelligence that has produced volumes of material [Ref. 2,6,8,11]. It is inappropriate to review all of the literature within the confines of this work. Rather, we begin with the generally agreed upon observation that, "State space representations are more suited to problems in which the final solution can be specified as a path or as a single node [in a graph]" [Ref. 6, p.26]. Clearly, a solution to the weighted-region problem is a path and thus, the state space representation is appropriate.

A state space representation of a problem includes both states and operators. A state is an encoding of the current progress towards reaching the solution. The operators specify methods of moving between states. Framing these definitions in the context of a path-planning problem, suppose that the task is to find the shortest-distance on-road route from intersection A to intersection B and that the system has determined the best path from A to an intermediate intersection, I. Figure 1 depicts this situation; the heavy line represents the path from A to I.

The state at the point depicted in Figure 1 describes progress towards the goal. Thus, it must capture the distance already traveled from A to I, the fact that the known path terminates at I, and some estimate of the cost remaining to complete the path from I to B. There is a single operator that allows movement from the current intersection (I) to those intersections adjacent to it (I1,I2,I3, and I4). If each intersection is labeled with its state information, then Figure 1 satisfies the definition of a *state space graph,* a graph where each state is connected to its successor states. In general, a state space graph includes one link for each operator that can be applied to any state.

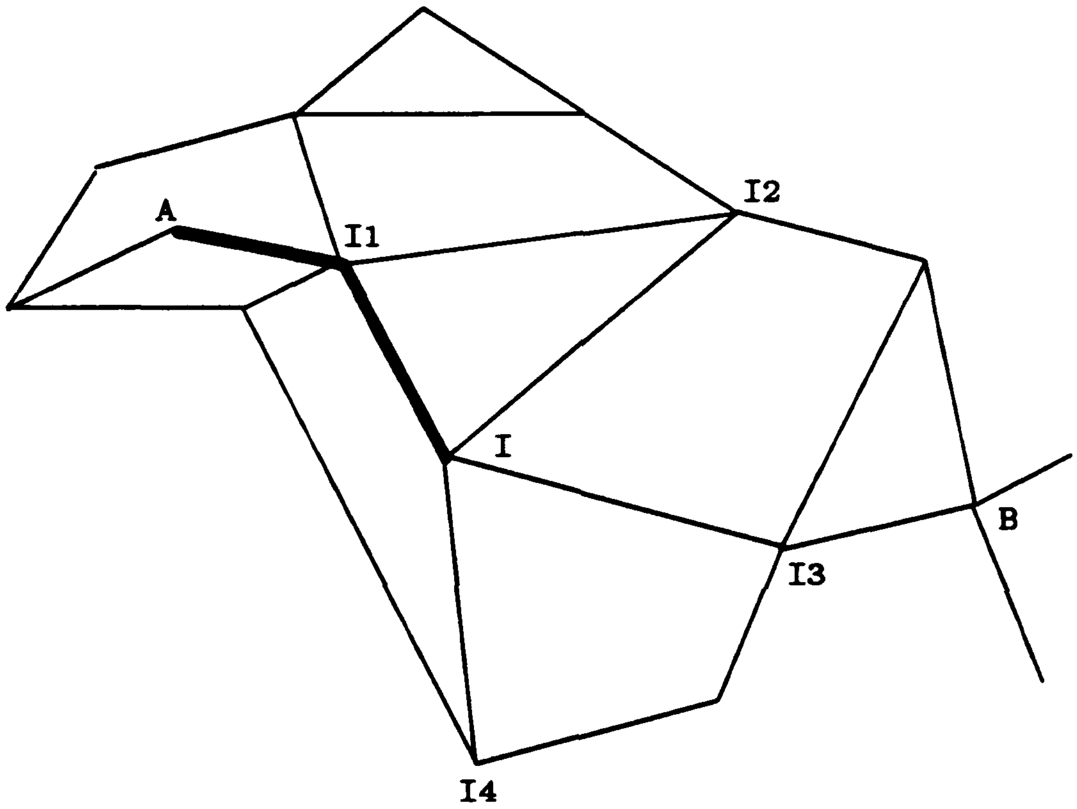


Figure 1. A Road Network Problem

Recall the requirements of the A* algorithm and note that these are all met by our definition of a state description. A* is an informed systematic strategy that attempts to avoid expanding useless states (states not on the solution path). Thus, A* *prunes* useless states from the search. Note that this ability is closely tied to the state description provided by the problem representation.

Due to the pruning ability evinced by the A* strategy it is classified as a *split-and-prune* [Ref. 6] method. In the split-and-prune paradigm, partial solutions (such as the path from A to I in Figure 1) represent a set of complete solutions (all paths from A through I to B). Whenever a partial solution is refined by applying an operator, yielding another partial solution, the set of possible solutions has been split. As an example, extending the partial solution of Figure 1 to intersection I3 splits the set of possible solutions from I into one subset that includes I3 as the next intersection on the path and one subset that does not. Then, those subsets representing solutions that cannot possibly contain the optimal solution can be pruned from the search.

Systematic strategies are complete and non-redundant. The completeness requirement implies that no set of possible solutions can be pruned if that set might contain the optimal solution. The non-redundancy requirement implies that splitting a set of solutions should not regenerate previously discovered partial solutions. Thus, a state space problem representation must account for these requirements. There is an obvious solution when the problem requires finding the shortest distance route on a road network. A correct representation for the weighted-region problem is not as self-evident.

A suitable problem representation for the weighted-region problem minimizes information loss, specifies easily computable state transformation operators, and supports the use of the split-and-prune paradigm by a systematic strategy. These are general requirements of the representation. More specific properties are developed in the following sections.

D. PLANNING

Research into the planning process has been a central activity in the artificial intelligence community for many years [Ref. 12]. The concerns usually involve task or activity planning while specific route-planning problems have received more attention from the operations research community [Ref. 13.14]. However, the operations research effort has generally been directed towards devising search strategies to be used on graph representations of static linear media, road networks as an example. Artificial intelligence work has been more focused on devising intelligent problem representations that enhance the search process. The weighted-region problem seems ill-suited to description by finite graphs since a continuous real-world environment must be modeled. Representing a continuous environment by a finite graph is tantamount to developing a 1:1 mapping between the reals and integers. Clearly, any such mapping cannot be totally information preserving. A principal goal then is to devise a problem representation that minimizes information loss while providing for efficient operators. In the following sections, we examine several planning systems that exhibit properties useful in solving the weighted-region problem.

1. The General Problem Solver (GPS)

GPS was developed as a model of the human problem solving process [Ref. 15]. It was intended to be sufficiently general so that it could be applied in a variety of planning domains [Ref. 12]. Comprehensive studies have been made of GPS performance in the areas of cryptoarithmetic, formal logic, and chess playing. Protocol analysis of humans solving problems in these same domains show that human behavior and GPS behavior correlate to a high degree [Ref. 12].

The main problem solving strategy incorporated in GPS is *means-ends analysis*. The planning process is viewed as an iterative application of operators that transform the start state into the goal state. The sequence of operators eventually used to produce the desired transformation becomes the finished plan. Operators are rated prior to application by measuring the amount that they can decrease the difference between the start and goal state. Means-ends analysis attempts to eliminate particular levels of difference through recursive techniques.

There are several drawbacks in the means-ends analysis paradigm. The heuristic nature of the method (rating differences and operator applicability) can create long chains of problem solving steps that abruptly dead end. Also, means-ends analysis does not strive for optimal solutions; any solution will suffice. Further, means-ends analysis does not deal with interacting subproblems effectively. If the completion of one subgoal prevents completion of another, GPS can only return to the start and attempt a different ordering of subgoals. Further, GPS requires specific knowledge allowing problem decomposition (by stating the effects of operator application). Thus, although GPS is a reasonable model of the human problem solving process in at least three domains, it does not seem applicable to this particular path-planning problem.

However, some key issues involved in the GPS paradigm are desirable. GPS presents a method for saving as much of a workable plan as possible, recursion. In fact, recursive decomposition will prove to be useful in solving the weighted-region problem. Rating the differences eliminated by operators and choosing for application those that produce the most substantial reductions is akin to the human ability to view the physical world as homogeneous regions, not as discrete points. GPS also utilizes a dynamically changing world view (again through recursion), not a static representation. Finally, GPS completes its plans in a hierarchy of abstractions.

2. Opportunistic Planning

In [Ref. 16] a different and more complicated theory of human planning is developed. This paradigm, called island driving or opportunistic planning, incorporates the human trait of attempting to produce optimal (in some sense) plans. This model recognizes the human ability to move freely between many levels of abstraction during the planning process. This characteristic has been termed multidirectional processing. Another salient observation of the human planning process is that it is opportunistic. Humans are able to exploit situations in which the completion of one subgoal is greatly enhanced by (or even included in) the completion of another subgoal. (If I go around the mountain, I'll not only avoid a long slow climb, but I'll end up very close to a major highway as well.) Through opportunism, human planning has a bottom-up component. Steps in a plan can be included when they are conveniently introduced.

The findings of [Ref. 16] include several key observations. Knowledge of the domain is very important and optimal planning relies on the specific capabilities (options) of the subject for which the plan is being developed (much as the area-cost map, as defined in Chapter I, is keyed to the abilities of a specific agent). Key observations about the physical world can dictate immediate inclusion of some parts of a plan (the river must be crossed at the bridge). This indicates the value of eliminating fixed steps in the planning process, such as dogmatically proceeding from start to goal in a graph search. A multidirectional approach can be beneficial. Finally, humans use levels of abstraction in planning. This suggests that representing terrain in a similar fashion can be useful. A simple abstraction might view terrain as homogeneous regions at one level and as lattice points at another.

The implementation of the [Ref. 16] model is quite complex and includes many different planning specialists who communicate through a shared blackboard that has five different planes of planning decision categories. The implementation is intended to serve as a general problem solving apparatus. Without further discussion, we suggest that the geometric nature of the problem at hand favors a less complex system. However, key observations made in [Ref. 16] should prove very useful.

3. Refinement of Skeletal Plans

The human problem solving process has also been studied and reported in [Ref. 17]. The key observation of this work is that humans tend not to "reinvent the wheel". Specifically, the study observed scientists who were planning experiments. It was noted that the process often began with an abstract proven skeletal plan that included the basic steps that the particular experiment should follow. Using domain knowledge of the specific problem at hand, one of a set of such general purpose plans would be expanded to produce the desired experiment plan. Thus, the general process can be viewed as the incremental refinement of a general-purpose skeletal plan.

The theories developed in this work have been implemented in the MOLGEN system that can be used to plan experiments in the molecular genetics field. MOLGEN has two primary components, one that chooses an appropriate skeletal plan, and one that refines this chosen plan. Work has not progressed far on the plan selection process. A table look-up of a taxonomy of predefined plan utility values comprises the general methodology. The refinement process relies on a large hierarchically arranged knowledge base of laboratory techniques. The steps of the selected plan are linearly processed against the knowledge base material to complete the process of plan instantiation.

This work formalizes an important human planning trait. Utilization of skeletal plans is the paramount example of reusing already expended effort. In a sense, this is a form of opportunistic planning. One can make use of a known solution to a similar problem to guide the search for a solution to the problem at hand. There may be a method to incorporate this technique into a solving system for the weighted-region problem. However, some difficulties must be overcome. The linear instantiation methods used by MOLGEN conflict with the interacting subproblems inherent in the weighted-region problem. Further, describing a previous solution by its general utility seems inappropriate. In cases where a problem is a nearly exact copy of a previous case, the applicability of skeletal plans is obvious. However, due to the continuous nature of the physical world, such a situation does not occur with any regularity. When new instances of the weighted-region problem are different from stored solutions to old problem instances, it is difficult to see how the old solutions might be exploited.

4. Nets of Action Hierarchies (NOAH)

The NOAH system was created to operate in the problem solving and planning domain [Ref. 18]. NOAH is regarded as the archetypal hierarchical planning system. This work is not based on the human paradigm per se, but is intended to address some of the key difficulties that were apparent in other planning models (such as GPS, HACKER, and INTERPLAN). NOAH plans actions in a procedural net framework that contains both declarative and procedural knowledge. The procedural knowledge is tied to a specific problem domain and is capable of expanding goals into subgoals. Declarative knowledge is used to express the effects of executing parts of a plan. Such knowledge is useful in noting how the state of the problem has been altered by executing particular problem solving operators.

A key contribution of NOAH was the use of a least-commitment strategy to avoid the difficulties of subproblem interaction suffered by most other planning systems. The least-commitment principle requires that subgoals not be ordered until absolutely necessary. Subgoals are assumed to be executable in parallel unless interaction difficulties become apparent. This philosophy may be applicable to the weighted-region problem. Since the weighted-region problem does not seem to be readily decomposable, intermediate subgoals may have to be selected at the latest possible time. NOAH includes active agents, called critics, whose specific task is to find interaction problems. Only when a critic reports a conflict is an ordering imposed on subgoal completion. As a trivial example, consider a movement problem involving a bridge. Suppose that the original problem is to move from start, A to goal, B, and that a river with a single bridge, located at C, lies between them. Domain (procedural) knowledge would decompose the original move(A,B) goal into two subgoals; move(A,C) and move(C,B). Declarative knowledge would be used to describe the problem state associated with each subgoal. A critic would then find the readily apparent subgoal conflict and suggest an ordering of first move(A,C), then move(C,B).

The key concept that can be learned from NOAH (relative to the general weighted-region problem) is the importance of actively planning for interaction conflicts. Delaying the selection of intermediate subgoals can lessen the impact of these conflicts. Also, knowledge plays an important role in this system. Both domain and declarative knowledge have been employed. Clearly, the value of domain knowledge has been espoused by most of the schema examined. NOAH has also shown the importance of declarative knowledge in reasoning about the state of the system itself.

5. Summary

We have seen that human-like planning can be a very complex process which need not be as well ordered as a graph search from start to goal. The amount of knowledge brought to bear on a problem is very important, not only in allowing decompositions, but in taking advantage of randomly occurring opportunities as well. Humans also tend to reuse frameworks of plans that have proven successful in past enterprises. Reusing known solutions to solved instances of the weighted-region problem may prove helpful. Rating the performance of available operators by the amount of progress (difference reduction) they achieve is also an important aspect of human planning. Actively planning for interaction conflicts is valuable. When such conflicts are found at an early stage, the number of subgoals involved will remain low and replanning will be minimized. Finally, levels of abstraction are also important in that they can simplify reasoning. The selection of an appropriate problem representation is fundamental in fulfilling these requirements.

E. SOLVING A BINARY-TERRAIN PATH-PLANNING PROBLEM

1. Introduction

Much has been accomplished in the areas of planning motions for robot manipulators and planning movements of mobile robots within localized areas. In both problem domains, the task is to plan an optimal-cost path for movement of the robot (or a robot manipulator) that does not intersect any obstacles in the physical environment (i.e., the physical working space for the robot). These problems are often termed *binary-case* problems because there are only two possible classifications for any point in the environment. In binary-case problem representations, every specifiable point in the environment is classified as traversable or non-traversable: thus every point belongs to a "free space" area or an obstacle area.

The binary-case problems constitute a simplified special case of the general weighted-region problem. Since all traversable space is the same, there is no point in including a cost term in the objective function that is to be minimized.

Removing the cost term simplifies the task to that of finding the shortest-distance path between the two given points, the start and goal. Thus, binary-case techniques take advantage of the *straight-line principle:* a straight line between any two points constitutes the shortest distance between those two points.

Occasionally, the solution techniques are designed to prefer paths that do not "run too close" to obstacle areas. These techniques allow for some amount of error by a robotic agent executing the plan because they view their plans as exact specifications for sequences of motion. A solution path for the general weighted-region problem normally covers a much greater range of motion than those plans generated by binary-case techniques. Thus, solution paths for the general weighted-region problem are more appropriately viewed as general guidance for movement, not as exact routes to be carefully followed. In this case, producing plans that allow for agent error is not a relevant concern.

Aside from this issue, solution methods for the binary-case problems solve a special case of the weighted-region problem. In both domains, the cardinal task is that of planning an optimal-cost route over a continuous space. Thus, examining the binary-case techniques is important. The most conceptually simple binary-case strategy is based on the generate-and-test paradigm. In this paradigm, a *plausible move generator* proposes a possible solution that is inspected for acceptability by a *test procedure.* [Ref. 8].

2. A Simple Localized-Improvement Model

The simple localized-improvement technique used to solve binary-case path-planning problems posits a representation of the environment that includes rectangular obstacles defined by the Cartesian coordinates of each vertex and of the starting and goal locations [Ref. 19]. The straight-line principle is used as the path generator in that a line segment from the start to the goal is proposed as the initial solution for the optimal start-to-goal path. This line segment is inspected for intersections with any of the rectangular obstacles. If no intersection is found, the problem has been solved. However, if an intersection does occur, intermediate subgoals defined by two of the obstacle vertices are proposed. The two vertices chosen are normally diagonally opposite each other (although it is possible that the two vertices define a face of the obstacle in some cases). Two new possible solution paths are then generated. One runs from the start to the first intermediate subgoal to goal and the other from start to the second intermediate subgoal to the goal. The corresponding line segments are then inspected for obstacle intersections. The results of such inspections dictate actions analogous to those required for the originally proposed path.

This method has several key advantages, the first of which is its conceptual simplicity. The required algorithms are easily implemented and do not require sophisticated techniques. Secondly, the method can utilize the straight-line principle to generate only plausible solutions. Obstacles and paths that do not intersect the generated (possible) solution paths and are thus far afield can be ignored. Thirdly, the method allows the problem description to be developed dynamically. The routes generated can be viewed as paths through a graph where graph nodes are obstacle vertices and graph links represent clear (non-obstructed) paths between obstacle vertices. The graph is created dynamically in that new nodes are added to the graph only when required by a collision (i.e., a path intersection with an obstacle boundary). A final advantage of this method is due to the combined effects of the first three characteristics. The method provides a solution relatively quickly because it avoids wasted computations.

This simple technique also has several distinct disadvantages. While modeling obstacles as rectangles allows the development of simple decision criteria for defining intermediate subgoals, it is a poor assumption to posit that every object in the physical environment can be adequately modeled by a rectangle. (One could argue that calculus is based on a similar premise. A key difference in this case is that a rectangle width can never reach zero as a limit and as rectangle width decreases, the number of obstacles to be considered grows accordingly. Such growth invalidates the simplicity and efficiency of the method.) A second, and much more serious, deficiency associated with this simple technique is that it may yield non-optimal solutions. The method functions perfectly if only one obstacle is involved. However, when two or more obstacles are present in the environment, interacting subproblems can confound the technique. Figure 2 shows an instance where this anomaly occurs.

In Figure 2. the original straight line path from start (s) to goal (g) is hypothesized by a plausible move generator. A collision on this path is detected and the interesting points of the obstacle are determined to be o1 and o2. Then, segments s-o1. o1-g, s-o2 and o2-g are tested. Segments s-o1, o1-g and s-o2 are found to be obstacle free. Another collision involving interesting points o3 and o4 is detected on the proposed o2-g segment. The new segments o2-o3, o3-g, o2-o4 and o4-g are all tested and determined to be obstacle free. Path lengths are then computed and the s-o1-g path has the shortest length of all paths, so it is

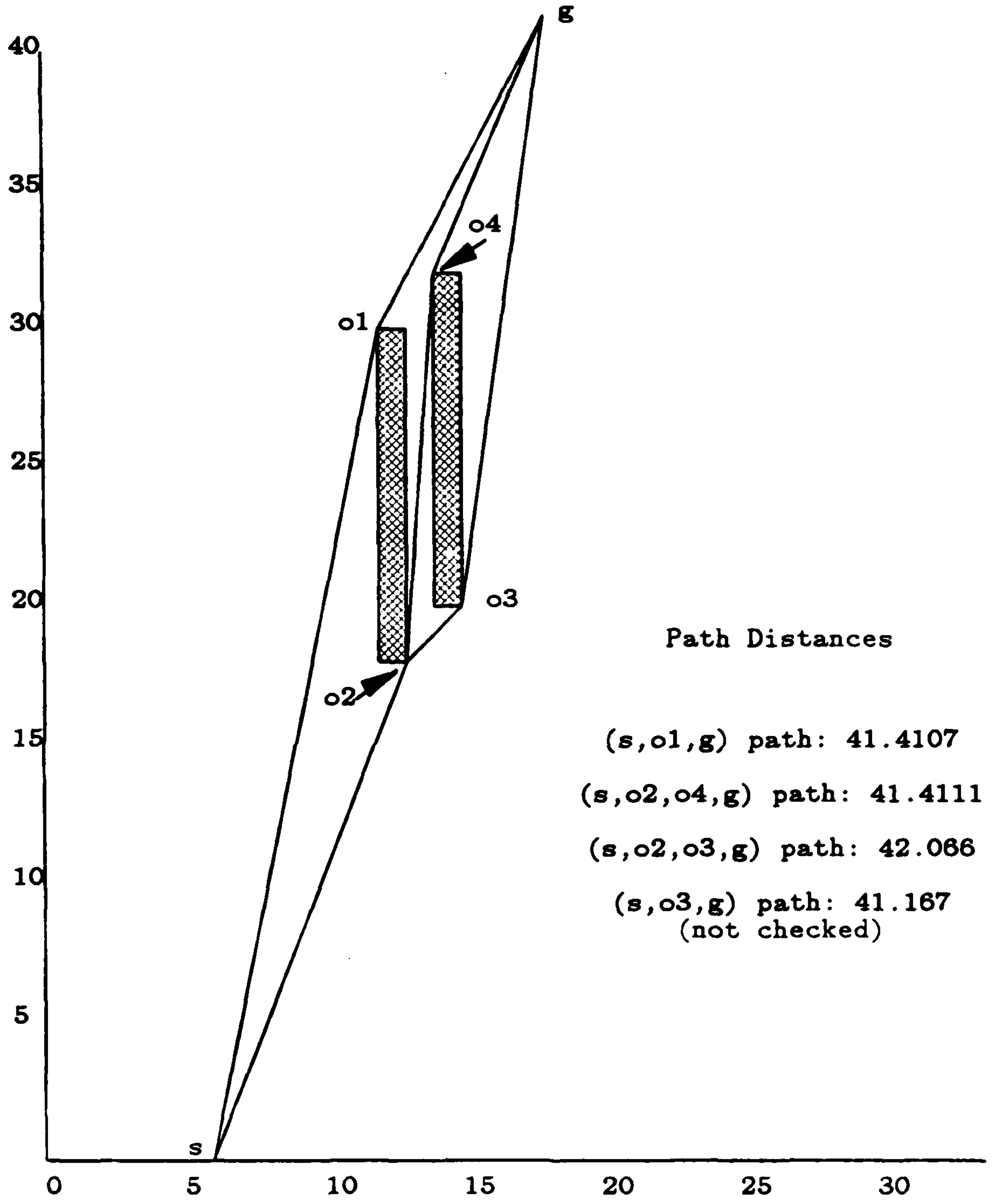


Figure 2. Fallacy of Simple Methods

returned as the optimal solution. However, the segment s-o3 is obstacle free. Thus, the route s-o3-g is legitimate and is clearly shorter than the s-o2-o3-g route that was used to determine optimality. It could easily be true that s-o3-g is shorter than the s-o1-g) route that was selected as the optimal-cost solution path.

From this example, it is clear that the simple and intuitively appealing localized-improvement approach may not yield optimal results. An additional procedure to smooth out (i.e., remove unnecessary points) the routes must be interposed between route finding and selecting the solution path if optimality is desired. The smoothing operation can be very expensive, dependent upon path length, because all points on the path, except adjacent points, have to be inspected for possible elimination. Therefore, a path with n points would require that $\sum_{i=1}^{i=n-2} i$ point pairs be inspected (an $O(n^2)$ operation). Moreover, all paths can be subjected to this procedure. Notice that in the example, the optimal path is generated by smoothing out the greatest-distance path.

3. The VGraph Solution Technique

An alternative to the simple localized-improvement model involves the search of an explicit undirected graph and has been called the Visibility Graph or VGraph model [Ref. 17]. Here, obstacles are modeled as convex polygons, represented by listing the coordinates of each vertex. The coordinates of the start and goal are also known. A graph is created such that each node in the graph corresponds to the starting coordinates, the goal coordinates, or the coordinates of an obstacle vertex. A link is included in the graph for each straight line segment that can connect any two vertex coordinates (represented by graph nodes) without intersecting an obstacle. Once such a graph has been constructed, standard graph search techniques such as Dijkstra's Algorithm or one of the A* family can be applied to find an optimal solution. No smoothing operations are ever required. The lower half of Figure 3 depicts a completed graph construction, built from the associated environmental space description (i.e., the configuration of obstacles in the working space of interest) shown in the top half of the figure.

The VGraph approach eliminates the difficulties associated with the simple localized-improvement paradigm. Obstacles can be more realistically modeled as convex polygons and truly optimal paths (in terms of the problem representation) are provided without the use of expensive ancillary operations. Also note that the straight-line principle is used in this method to determine membership in the link set of the graph. However, there are also costs associated with the VGraph method. The creation of the problem representation can require a large amount of computation. If there are N nodes in the graph then $\sum_{i=1}^{i=N-1} i$ line segments must be inspected for possible inclusion into the link set (an $O(N^2)$ operation). (However, this is an absolute upper bound on inspections. An intelligent procedure would note that interior chords produced by two vertices of the same obstacle need not be inspected. However, if obstacles are allowed to overlap, the number of inspections required tends toward the upper bound.) Thus, unlike the localized-improvement method, the VGraph method considers all obstacles in the environment in that the graph is statically created before search begins by an exhaustive search of all obstacles. In Figure 3 for example, all links associated with obstacle A are extraneous. In a situation where path planning takes place

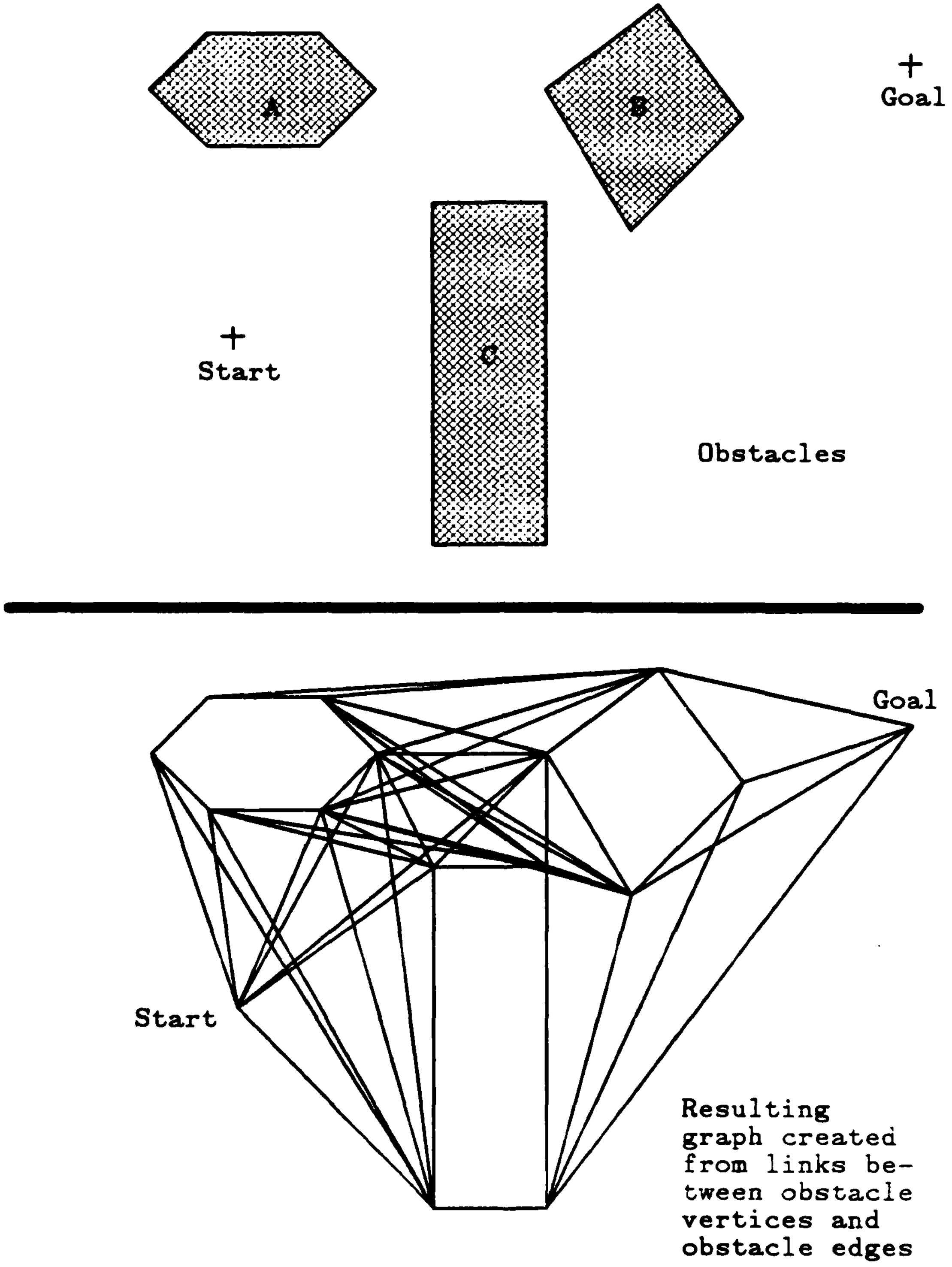


Figure 3. VGraph Obstacle Space and Graph

over the same area many times, the costs of graph creation can be spread out and thus made less important over time. However, such a situation may not always prevail.

4. Free Space Characterization Methods

An approach that relies on the use of generalized cones [Ref. 20] developed in connection with vision research has also been developed. Generalized cones (also known as generalized cylinders) are normally used to represent the volume and shape of three-dimensional objects. The cones are described by sweeping a cross-sectional area (two-dimensional) along a curve in space called a spine. The shape of the cross section is deformed by a predefined sweeping rule as it moves along the spine [Ref. 21]. As a path-planning paradigm. the free space between obstacles is described as a series of overlapping, two-dimensional cones. The cones have straight spines and the cross sections are represented as line segments. This explains the loss of one degree in dimensionality since line segments are used as opposed to areal figures. The line segments (used in lieu of cross sections) are positioned perpendicularly to the spine and the length of the left and right portions of the segment are varied independently as sweeping takes place. The sweeping rule is a predefined piecewise-linear function created by measuring distances to obstacle edges from the spine [Ref. 21] (see Figure 4). Any two obstacle edges are candidates for creating free space defining cones. Essentially. the requirements for two edges to define a cone are that they belong to different obstacles and that they approximately face each other. (Obstacle edges face each other when they lie approximately parallel and they have no other obstacle edges separating them.) Once a complete set of cones has been formed, a graph is

Dashed lines indicate cone spines
Shaded areas are obstacles

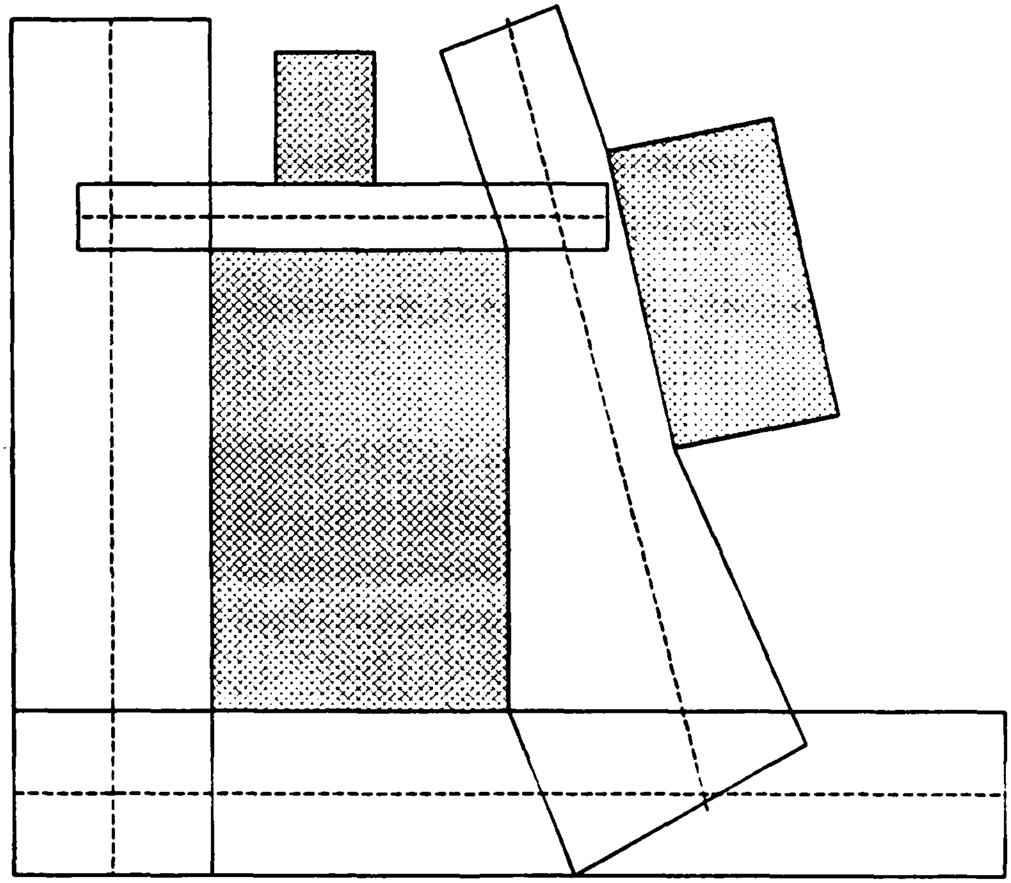


Figure 4. Generalized Cones of Free Space

constructed by using spine intersection information. The nodes in the graph correspond to the coordinates of spine intersections. The graph link set is composed of links between consecutive spine intersections. The graph is essentially a Voronoi diagram [Ref. 14] of the environment space. A solution to the path-planning problem is provided by conducting a search of the graph representing this Voronoi diagram.

The cost of this algorithm can be high, primarily due to the graph creation. If E is the number of obstacle edges, then the time complexity is, at worst $O(E^4)$, but may be as low as $O(E^2 c^{2\sqrt{\log n}})$ due to the similarities to the Voronoi diagram [Ref. 21]. This free space characterization method is primarily concerned with lessening the rotational problems (i.e., how to rotate an irregularly shaped body so that it can "fit" between slightly separated obstacles) and the "not too close" problem for a two-dimensional object (i.e., not a point) moving through a two-dimensional space and thus has added complexity. However, it is interesting in this analysis for several reasons. As usual, the method employs the straight-line principle and attempts to establish a graph-theoretic basis to facilitate the search for the optimal path. A new feature is that a free space characterization is emphasized, not an obstacle space representation. Emphasis on free space may be important for a general-case solution because it is the diversity of these traversable areas that gives rise to much of the added complexity of the problem. when compared to a binary-case representation. Also, characterization of free space results in a smaller graph than that required by the VGraph method. This method has the salient drawback that following spines does not produce optimal routes.

5. The Potential-Fields Method

Another binary-case solution technique is called the potential-fields approach [Ref. 22]. This technique is also concerned with planning for paths that do not run too close to obstacles. Conceptually, obstacles are modeled as areas of increased elevation, i.e., hills with sloping sides where the hilltop is the center of the obstacle. The object to be moved can be regarded as a ball bearing that has an initial location corresponding to the starting coordinates. In operation, the entire environment space is "tilted" from the start to the goal so that the ball bearing "rolls" in the desired direction. The path followed by the ball bearing is provided as a solution.

This method has some salient deficiencies. The ball bearing can roll into a box canyon and become trapped before arriving at the goal. In such an instance, backtracking measures are necessary to restart the procedure. Also, a path that requires going over a small rise near the start is avoided, even though it may lead to the optimal solution path.

The potential-fields method employs a graph-theoretic basis in the form of a regular grid. The straight-line principle is also brought to bear in the form of gravity. Thus, key similarities are present when compared to the other methods. In many ways, the method is similar to the wavefront-propagation technique (discussed in Section II.F.2). The potential-fields method models a continuously varying cost (i.e., elevation change) as discrete point costs in a lattice-like graph.

6. Summary

There are other methods which have been developed to solve the binary-case problem [Ref. 23.24,25,26,27.28,29,30]. Although differing in detail, each of these can be classified as a version of one of the methods discussed above. Important problem characteristics can be extracted from the examination of solution techniques. Interacting subproblems can confound problem decomposition, as evinced by the simple localized-improvement method. The VGraph method used an exhaustive search of a particular representation (the visibility graph) to overcome the interaction problem.

A primary strategy is to find a graph-based problem representation, either by the characterization of obstacles or free space. Once such a representation has been established, well-founded graph search techniques can be applied to solve the path-planning problem. However, the creation of an exhaustive graph can be computationally expensive. A dynamically created graph is more efficient (as in the simple localized-improvement model) if interaction problems can be overcome. A problem with the dynamic graphs of the simple localized-improvement method is that only local information is used. This *greedy* method is insufficient to support the requirement for global optimality. Finally, the straight-line principle is crucial to the success of each method. Many of these observations are important in developing solutions to the general-case weighted-region problem because of shared characteristics. In fact, one solution method, known as the dynamic programming, regular grid or wavefront-propagation method, has often been applied to both versions of the weighted-region problem. We now examine solution techniques appropriate for the general-case weighted-region problem.

F. SOLVING THE GENERAL PATH-PLANNING PROBLEM

1. Introduction

The general-case weighted-region problem differs from the binary-case version by the inclusion of cost ratings for the traversable areas. As the cost of traversing one area can be quite different from another, a new parameter is introduced into the cost computation for each possible path. In the binary-case problem, the cost of traversing every route is computed by the simple formula $Cost = \sum_{i=1}^{n} d_i$ where there are n line segments in the complete path and d_i is the Euclidean distance along the ith line segment. In the general-case problem, the formula becomes $Cost = \sum_{i=1}^{n} c_i d_i$ where n and d_i have the same meaning. The new parameter, c_i is the cost per unit length of the ith line segment. Also, the number of line segments is typically increased. As an example, a path crossing two different cost areas consecutively is represented by two line segments. Only one line segment is required to reflect the same situation in a binary-case representation. The addition of cost information has the effect of invalidating the straight-line principle so prevalent in the binary-case solution techniques.

A second major difference between general-case and binary-case instances of the weighted-region problem is based the opportunity to perform problem decomposition. In the binary-case version, decomposition can be done because the optimal-cost start-to-goal path must either be a straight line between them or a path that includes obstacle region vertices as intermediate turn points. Thus, it is simple to construct a graph representation of the problem, as done in the

VGraph technique. In general-case instances of the problem, decomposition into subproblems is very difficult to achieve. There are some special cases, (see [Ref. 31] for example) where a graph similar to that used in the VGraph technique can be constructed. However, in general, it is not possible to specify a finite set of points that must include all turn points on an optimal-cost start-to-goal solution path. Because of this, the general-case instance of the weighted-region problem is much more difficult to solve. Thus, there has not been as much progress in solving the general-case weighted-region problem as we have seen for the binary-case version. The first attempts to solve the general-case problem were extensions of an exhaustive search method used to solve strict binary-case representations.

2. Wavefront Propagation

Wavefront propagation is the most commonly used method to solve the weighted-region problem. The method's popularity is justified by its conceptual simplicity and flexibility. The technique can be applied to both binary-case and general-case instances of the weighted-region problem without modification in either implementation or computational complexity. (Of course, the problem representation must reflect the task at hand.)

Wavefront propagation can be viewed as an extension of the VGraph philosophy in that the complexity of solving the problem is primarily borne by some component that creates the specific graph-based problem representation. Given an appropriate representation, a standard weak method of search is applied to yield an optimal-cost solution path. When using wavefront propagation, the graph-based problem representation is actually a lattice structure, as described below.

a. Basic Wavefront Propagation

Recall that the VGraph technique relies on an exhaustive graph consisting of a node for each obstacle vertex and a link for each pair of vertices that can be connected by an unobstructed line segment (i.e., pass a "line of sight" test). This structure is appropriate for binary-case problems because of the straight-line principle. However, the same graph structure is not valid for the general-case weighted-region problem because region vertices do not play the same all-important role. Turn points on the optimal path can occur anywhere. However, if the graph consists of a node for every representable point in the environment, then the graph must include a node for every representable turn point. The *resolution* of the problem representation plays an important role here. The number of discrete points used to model the continuous environment specifies the resolution of the representation. The nodes are uniformly spaced and resolution determines the intra-node spacing. This is the basic premise of the wavefront strategy; apply a standard search strategy to a finite exhaustive graph.

Having established the basic premise, the wavefront strategy departs from the VGraph model. Instead of creating an explicit link between each two obstacle vertices having line of sight, the wavefront graph implicitly includes a link between each node and all of its physically adjacent neighbors. So, instead of representing long distance, unobstructed path segments, the wavefront graph links represent small movements that could be made from each discrete point in the environment modeled by a graph node. Thus, there is a regular pattern of links between uniformly placed nodes. The result is as if a grid (or lattice structure) has been superimposed on the environment. The graph is made applicable to the general-case problem by associating a variable cost with each link, reflecting the cost to traverse space in the environment represented by the link. (For a binary problem, the cost is constant.) Nodes that lie in obstacle areas have conceptually infinite-cost links.

Again, resolution plays an important role. A determination must be made as to how many physically adjacent nodes should be recognized as neighboring since this determines the branching factor at each node. The more links in the graph, the more time required to search the graph and the more space required to store the graph. Specifying eight neighbors is usually judged as the point of diminishing returns between time and space required to reach a solution and the accuracy of the solution. To understand this, consider the operation of the strategy.

Wavefront propagation applies omnidirectional, uniform-cost search (as defined in Section II.B.1) to a directed graph representing the environment. It is essentially a dynamic programming solution to the problem. Recall that uniform-cost search finds the optimal path to each node in the graph that can be reached before the solution is found. This is the dynamic programming principle of solving all subproblems in order to solve the overall problem. In a physical analogy, the wavefront-propagation process is akin to dropping a pebble into a calm body of water and observing the propagation of the resulting wavefront. When the wavefront touches the goal, a solution path can be retrieved by tracing gradients to "snapshots" of the wavefront back to its origin (which corresponds to the start location). Implementations using uniform-cost search reduce the gradient-tracing requirement to referencing backpointers set as the search progressed.

Resolution is important to the accuracy of the solution paths reported by wavefront methods. A true wavefront (as in the pebble and water analogy) would change its position along a continuum. Implementations of the wavefront-propagation algorithm model this change of position as a series of discrete points (nodes in a graph). Obviously, the number of nodes in the graph helps determine the accuracy of the solution. Thus, the satisficing nature of wavefront propagation becomes apparent; a simplified problem representation is used to reduce the amount of search required to arrive at a "satisfactory" solution path. In general, however, the weighted-region problem is a semi-optimizing task because, in most cases, the true optimal path can only be described as two-tuples of real numbers indicating turn points of a path in the Cartesian plane. The exact real numbers cannot, in general, be represented on a finite-precision machine. Thus, because the task is semi-optimizing, the problem representation only needs to provide a resolution that ensures an acceptable level of error. Often, the choice is made to equate a screen pixel to a node. When a map is displayed on a computer graphics screen, the pixel is the highest possible unit of resolution. Thus, wavefront-propagation methods are sometimes referred to as *pixel planners*. The pixel resolution allows the satisficing nature of wavefront propagation to approximate, as closely as possible, the semi-optimizing character of the weighted-region problem.

The number of nodes in the graph is essentially a localized resolution issue. The number of links (per node) in the graph has a more global effect on solution accuracy in that the branching factor at each node determines the physical pattern of the search. As a simplifying assumption for illustrative purposes, suppose that the start is placed at the origin of a two-dimensional coordinate system, that the cost of traversing links is constant, and that no obstacles are present. Then, in a perfect situation where there is a link between each two nodes having line of sight (a conceptually infinite branching factor at each node), a uniform-cost search expands in a circular pattern (centered at the start). Constraining the number of allowable links modifies the search pattern by introducing approximations. That is, the circular shape is approximated by a (linearly sided) polygon inscribed in the circle. The polygon vertices lie exactly on the circle and are determined by sequences of homogeneous links (links that do not change direction). Thus, the number of polygon vertices corresponds exactly to the number of links allowed at each node. Between each polygon vertex, the circular shape is approximated by a chord.

Suppose that the problem is as described above and that the search relies on a four-way connected graph (i.e., a branching factor of 4 at each node) where each node has two vertical and two horizontal outgoing links. Then, the pattern of search (node expansion) during wavefront propagation assumes the shape of a square that has vertices on the x and y axis, as in Figure 5. The accuracy of a solution based on this model is worst where the chord is farthest from the circumscribing circle. This occurs on headings that are multiples of 45 degrees from the origin. For these points, uniform-cost search yields a solution path having cost C when the true straight-line path has cost TC, the Euclidean distance of the path multiplied by the appropriate cost factor. The error in the solution can be expressed as a ratio of the actual cost to the computed cost.

$$TC/C = cos(45) = 0.707$$

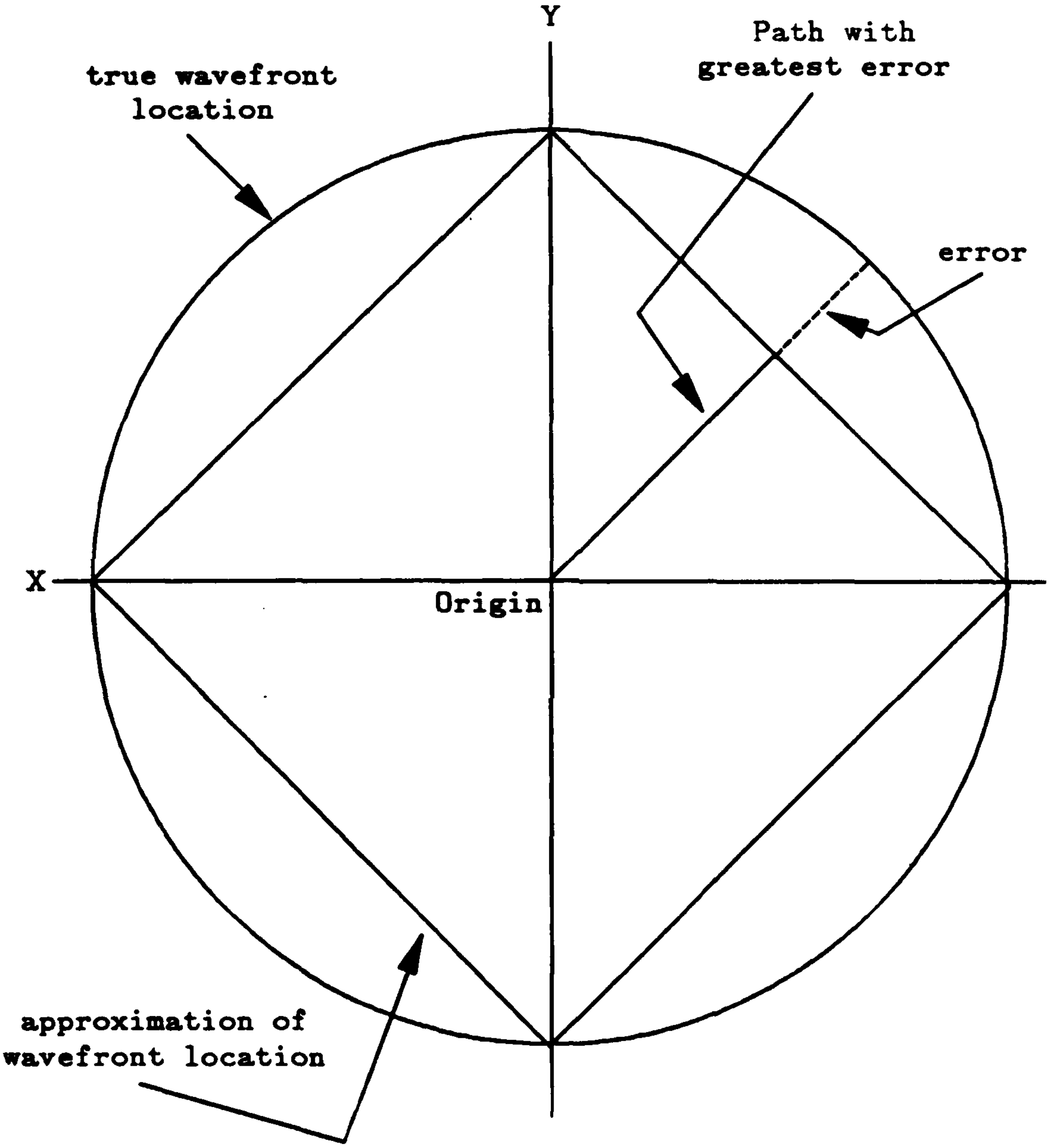


Figure 5. Wavefront Search Pattern

Thus, there is a potential for approximately 30% error in the solution cost. Increasing the connectivity of the graph so that diagonal links are also represented results in an eight-way connected graph. Again, the greatest error occurs at midpoints of chords, now located on headings that are multiples of 22.5 degrees from the origin. This representation limits the maximum error of path cost to approximately 8% and has become the practical standard of acceptable error for wavefront-propagation implementations.

The time complexity of wavefront propagation is best expressed in terms of the number of nodes expanded during the search. In the worst case, the search expands all nodes within a circle of radius equal to the cost of the solution path. The area of a circle is πr^2, so the complexity can be loosely tied to an $O(n^2)$ bound. Relying on this bound, it is clear that there is a direct tradeoff between the time and space required to solve the problem and the accuracy of the solution. The predetermined resolution fixes this tradeoff. Note that increasing the resolution by a factor of X increases the number of nodes in the graph by X^2. As an example, to represent a 10 square mile area using a resolution of 1/10 mile, 100 nodes required. Increasing the resolution by a factor of 10 to achieve a resolution of 1/100 mile necessitates 10,000 nodes.

b. A* Search and Wavefront Propagation

Work reported in [Ref. 32,33] has identified problems in the wavefront strategy and implemented partial solutions for their effects. *Digital bias* is an effect that is evident in wavefront solutions and is directly attributable to the discrete graph representation of a continuous environment. Specifically, wavefront solutions use connected series of line segments to model straight line paths, i.e., a "stair-step" approximation to the line. This modeling anomaly means that there is a set of optimal paths, all having the same digital cost, between almost every two points that can be named (the exception occurs when the optimal path consists of homogeneous links, as defined above). As an example, Figure 6 depicts three paths across a uniform-cost region that all have the same number of vertical and horizontal links and thus have the same digital cost. Clearly, the middle path in Figure 6 best models any single line segment.

The wavefront-propagation techniques reported in [Ref. 32,33] include heuristics that reward "corner points" (i.e., points where the path changes heading), thus favoring paths with more turn points. This strategy prefers the middle path of Figure 6 based on this heuristic. It was also noticed that using a lower-bound cost evaluation function, such as that required by A^* search, favors the desired paths. When Euclidean distance assumed to be traveled at optimal cost is used as the lower-bound component, the composite value (i.e., the f(N) rating) is lowest for those paths closest to a line between two points. These heuristics do not totally overcome the problems of digital bias. At first thought, it seems that A^* evaluations would defeat the problem. However, recall that the lower-bound function estimates remaining distance to the global goal, not to intermediate turn points along a path. Newer work has used *simulated annealing* [Ref. 34] as an optimization procedure to reduce the stair-step appearance of solution paths [Ref. 35].

Some work reported in [Ref. 32,33] also centered on using $A_\epsilon{}^*$ as a search strategy and performance improvements (in time) ranging up to twenty times are cited. It should be noted however, that this work is intended to support

Note that the paths have been offset vertically by two units (for clarity). All paths begin and end at the same locations and have equal lengths.

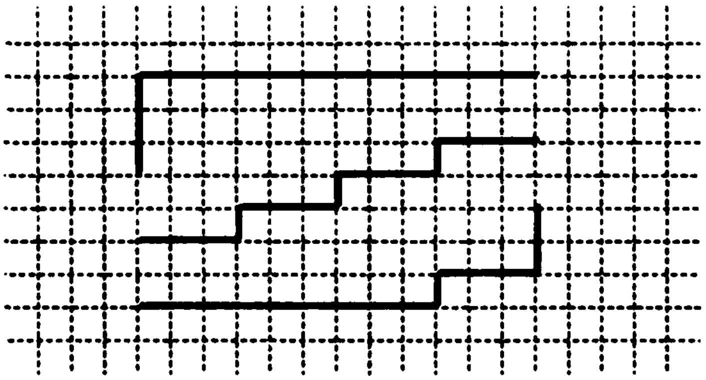


Figure 6. Digitally Biased Paths of Equal Length

a specific wheeled autonomous vehicle, the DARPA sponsored autonomous land vehicle (ALV) built by Martin Marietta [Ref 36]. The ALV is best suited to roadway travel and thus greatly prefers that media over all other terrain features. The link costs in the graph subjected to $A_\epsilon{}^*$ search correctly reflect this preference. As a result, the $A_\epsilon{}^*$ search generally resembles a standard wavefront until a roadway is reached. The search then proceeds along the road network until the goal is found. Off-road shortcuts are not considered. Clearly, the $A_\epsilon{}^*$ technique may not offer the same time improvements when the agent for which the path is being planned does not greatly favor one medium for travel over all others.

c. Wavefronts Exploiting Parallelism

There are several implementations of wavefront propagation that exploit the advantages of parallel-architecture machines [Ref. 29,36]. The most prominent development is the ADS system [Ref. 36], again intended to support the DARPA ALV [Ref. 36]. The work reported in [Ref. 36] refers to the strategy as a dynamic programming solution, which, as has been noted, is a correct characterization of the wavefront-propagation strategy.

Discussing the ADS system requires introduction of some new terminology. The graph used by a wavefront-propagation strategy can also be thought of as a cost map divided into a regular structure of small cells. Recall that the wavefront graph includes uniformly spaced nodes. Suppose that the nodes are drawn on paper so that their spatial arrangement reflects the physical displacement of the real world terrain points that they represent. Drawing in the arcs of a four-way connected graph results in a regular grid. This grid is equivalent to a set of regular cells. Instead of assigning costs to links, assign costs to each cell, reflecting the cost to move through the cell (in any direction).

The ADS system utilizes a conceptual structure similar to the map made of many small cells as described above. In addition to the stored cost for passing through each cell, they also use a *Figure of Merit (FOM),* an accumulated cost to reach a cell from a known starting point. Initially, the cell containing the goal point has a FOM of 0 and all other cells have infinite FOM's. The algorithm operates by selecting a cell and trying to replace the FOM's of neighboring cells, based on the FOM of the selected cell. A FOM in a neighbor cell is replaced if the FOM of the selected cell plus the cost to move into the neighbor cell is less than the FOM already stored in the neighbor cell.

Up to this point, the ADS implementation is essentially the wavefront-propagation technique based on a slightly different conceptual structure. However, note that in a graph having constant link costs, uniform-cost search is breadth-first search. In the ADS system, a depth-first component in added. The algorithm "sweeps" across the map in a specific direction (i.e., left to right, top to bottom, etc.). Each time a cell is selected, its eight neighbors are examined for possible FOM replacement. After the eight neighbors are updated, the one neighbor that corresponds to movement in the same direction (i.e., the right neighbor in a left to right sweep) becomes the new selected cell and its eight neighbors are examined for FOM update. This process begins at one edge of the map and continues moving in the selected direction until the opposite edge of the map is encountered. Once every row (or column, depending on the direction of sweep) has been swept across, the algorithm names a new starting edge and a new direction for sweeping. The map is swept in this manner, attempting to change FOM values on each sweep, until no cell changes its FOM value.

The ADS system exploits parallelism by assigning different processors to different "swatches" of the map. To illustrate, the ADS system uses maps that are 512 cells wide by 512 cells high. Suppose that the direction of sweep is left to right and that two processors are available to conduct the search. Each processor is assigned a 256 high by 512 wide "swatch" of the map to examine. FOM propagation occurs independently within the two swatches. Results reported by ADS state that the FOM values stabilize after 20 to 30 sweeps. ADS has published some timing results for the algorithm. Solving a problem on a (uniprocessor) DEC VAX 11/780 required 10 minutes. Solving the same problem on a Butterfly machine (see [Ref. 36]) with 40 processing units (computational nodes) required 1.05 minutes.

There is a side effect that arises from starting the propagation at the goal instead of the start. The ADS dynamic programming method yields the optimal path to the goal from every cell on map. If an agent strays off-course during the execution of a planned route and the goal has not changed, the agent need only locate itself in the correct map cell and retrieve the new optimal path. No further computations are required.

An attractive alternative to the ADS parallel wavefront-propagation implementation could be based on machines having mesh-connected architectures, such as the connection machine [Ref. 37]. The lattice structure problem representation used in wavefront propagation mirrors the physical structure of such machines. One computational element could be assigned to each node in the lattice and the physical connections in the machine could model links of the lattice. Theoretically, this organization would establish an $O(n)$ time bound for the algorithm (where there are n nodes in a solution path). Also, the wavefront-propagation technique has been implemented on neural-network machines in a similar manner, although no specific time requirements have been cited in this work [Ref. 38].

d. Linear Programming

It seems appropriate to mention the fact that four well-known problem-solving techniques have been mentioned. *Generate-and-test* was involved in the first binary-case solution method examined. In connection with the same examination of the localized-improvement technique, difficulties with interacting subproblems were discovered. This characteristic has serious effects on *divide-and-conquer* strategies. Many discussions have mentioned *greedy* techniques, such as the A* algorithm. The fallibility of total reliance on local information has been shown (in connection with the localized-improvement technique for example). Wavefront propagation falls into the fourth classical category of *dynamic programming* models since it solves all subproblems as a means to securing the single desired solution to the overall problem. For completeness, we note that the classic technique of *linear programming* used in operations research can also be applied to the weighted-region problem. Linear programming is discussed in connection with the wavefront-propagation models because both techniques rely on the same problem representation.

The graph used by wavefront-propagation techniques can also be viewed as a network. In this context, the weighted-region problem becomes the classic operations research problem of finding an optimal path by solving a minimum-flow through a network problem. In the minimum-flow problem, the start is the *source* and the goal is the *sink*. Link costs are the same. A single unit of flow is injected at the source and balance equations are used to force the flow out of the sink. Assume there are n nodes in the network and let the cost along the link from $node_i$ to $node_j$ be denoted c_{ij}. Assume that the source is $node_1$ and the goal is $node_g$. Then, the mathematical formulation is:

$$Minimize \sum_{i=1}^{g} \sum_{j=1}^{g} c_{ij} x_{ij}$$

Subject to

$$\sum_{m=1}^{g} x_{1m} - \sum_{n=1}^{g} x_{n1} = 1$$

$$\sum_{m=1}^{g} x_{gm} - \sum_{n=1}^{g} x_{ng} = -1$$

$$\sum_{m=2}^{g-1} x_{im} - \sum_{n=2}^{g-1} x_{ig} = 0$$

$$x_{ij} \epsilon [0,1] for i, j = 2,3,\ldots,g-1$$

The constraints restricting x_{ij} to be either zero or one are used to indicate those arcs on the minimal-cost path. The network flow formulation can be transformed into a simple (non-integer) linear programming formulation since the flow conservation equations require the problem to be unimodular. Thus, the constraint $x_{ij} \epsilon [0,1]$ can be simplified to $x_{ij} \geqslant 0$ and the standard simplex algorithm can be applied. Rewriting the simplified formulation yields:

$$Minimize \sum_{i=1}^{g} \sum_{j=1}^{g} c_{ij} x_{ij}$$

Subject to

$$\sum_{m=1}^{g} x_{1m} - \sum_{n=1}^{g} x_{n1} = 1$$

$$\sum_{m=1}^{g} x_{gm} - \sum_{n=1}^{g} x_{ng} = -1$$

$$\sum_{m=2}^{g-1} x_{im} - \sum_{n=2}^{g-1} x_{ig} = 0$$

$$x_{ij} \geqslant 0$$

This formulation has been included only for completeness of discussion. Since the weighted-region problem has only positive costs associated with each link, it falls into a special category of minimal-cost network flow problems. Thus the linear programming formulation is impractical. Least-cost path problems with non-negative link costs are more efficiently solved by other methods, notably the uniform-cost strategy employed by wavefront-propagation techniques [Ref. 13].

3. The Calculus of Variations Method

After the Second World War, significant importance was given to the problem of computing optimal trajectories for missile flight. Later, in the 1960's, the optimal routing of ocean-going ships was studied in a similar fashion. Both of these problems are similar to the general-case weighted-region problem. All three posit a starting location, a goal location, and the existence of forces that act against the movement of an object en-route. The forces are heterogeneous. A calculus of variations problem, formally known as the problem of Bolza, was successfully solved in the missile trajectory and ship-routing domains. Conceptually, the method operates by proposing an initial path from the start to the goal and allowing the prevalent forces along the route (the costs from the area-cost map in the weighted-region problem) to warp this path until it becomes optimal [Ref. 39].

This method is presented here because it has been used to solve problems similar to the general-case weighted-region problem without reliance on reduction to a graph theoretic basis. Instead, a complex and very powerful mathematical technique has been applied. (Calculus of variations develops a calculus for functions of functions. An introduction to the subject area can be found in [Ref 4]. A very brief overview is contained in [Ref. 40].) The calculus of variations approach is not totally appropriate for the weighted-region problem for several reasons. First, to avoid convergence on a local minimum, the method requires a reasonable approximation to the optimal solution as input to be used for an initial path. While this is relatively easy in the missile trajectory and ship routing domain, obtaining a fairly close approximation in the land route-planning domain requires effort tantamount to fully solving the problem. Without a given, reasonable initial solution, the calculus of variations method may never converge. Secondly, the method requires continuous derivatives of the active forces in the environment space. These cannot be guaranteed in the weighted-region problem. Also, solving the Bolza problem with the calculus of variations method requires a discrete representation of the environment, in that vectors of forces must be associated with discrete portions of the environment. Associated difficulties have been presented in connection with the wavefront method. The calculus-based method is not guaranteed to find a global minimum. Less expensive techniques can provide local minima for the weighted-region problem. The final difficulty associated with the calculus of variations model involves its computational costs. Recall that a primary difficulty of the wavefront method is its wastefulness and computational excess. It seems that the mathematical complexity of the calculus of variations model poses similar problems in terms of computational cost. There should be a simpler method to solve the weighted-region problem, based on the structure of the problem itself.

4. The Homogeneous Regions Model

A method for reducing the size of the graph used as the problem representation, which we term the homogeneous regions method, has been reported in [Ref. 41]. A key assumption of this method is that the physical terrain can be described as a finite number of large "patches", each of which exhibits uniform traversability characteristics. Archetypal "patches" are areas such as swamps or open fields. This organization is similar to that described in connection with the area-cost map of Section I.B. However, in [Ref. 41] these areas are assumed to be convex and centrally-symmetric so that the distance from the center of the representing polygon to any point on the polygon boundary is approximately equal. Given these characteristics, a graph is created where the nodes are patch center points and links connect all physically adjacent nodes. The cost of traversing a link is determined by finding the proportion of the link that lies in each associated patch and multiplying through by an appropriate fraction of each patch cost figure [Ref. 41].

There are several obvious difficulties with this method. Most real-world terrain does not seem to fit well with characterization by centrally-symmetric polygons. Linear features, such as roads and rivers, are prime examples of objects that are not easily characterized by centrally-symmetric polygons. Further, moving from area center to area center can produce errors which are difficult to estimate and thus the method, except in very rare cases, does not produce optimal paths. The method also espouses a beam-search strategy [Ref. 5]. Such a search strategy omits from consideration any feature that lies outside the selected beam width. No basis is provided for this assumption, nor is such a basis readily apparent. Thus, this method is truly a satisficing technique. A greatly simplified problem representation is used to reduce the search effort, resulting in solution paths that may not be optimal.

The important aspect of the homogeneous regions method is evident in its title. An effort is made to avoid the exhaustive uniform-grid representation of the environment by explicitly recognizing the fact that homogeneous-cost regions do occur in the real world. Although the proposed usage of this observation does not seem feasible, an important contribution has been made by stating the premise.

5. The Continuous Dijkstra Technique

In his PhD. dissertation, Joseph S. B. Mitchell develops an elegant method for solving the weighted-region problem [Ref. 42]. (Note that the work reported in [Ref. 42] was completed independently and simultaneously with the work reported in this thesis.) Mitchell's *Continuous Dijkstra Algorithm* (CDA) can be applied to three-dimensional binary-case or two-dimensional general-case instances of the weighted-region problem. In the latter instance, CDA relies on two key concepts. The first is that a homogeneous-cost region representation of the problem (similar to the area-cost map defined in Section I.B) is more appropriate than a graph consisting of uniformly spaced nodes and predetermined links. The second is that *Snell's law* can be borrowed from optics and applied as a basic guiding principle for local optimization in the general problem. Snell's law plays a fundamental role, similar to that played by the straight-line principle in binary-case problems.

Snell's law is used in ray optics to characterize the refraction path that a light ray follows when projected through optical media of different refractive indices. The relation expressed by Snell's law is possible because Fermat's Principle states that the optical path length along a light ray from some initial point to some terminal point must be an extremum [Ref. 43]. Without providing all the details necessary to adapt Snell's law to the weighted-region problem (which is the subject of Chapter IV), we note that there is a similarity between the two problem domains. Equating homogeneous-cost regions to optical media, the cost of passing through a region to refractive indices, and minimum-time paths to minimum-cost paths makes the similarity evident. Specifically, suppose that there is a flat sided, glass container partially filled with water and that a pencil is suspended in the container so that it is partially under water. Looking at the pencil through the glass, it appears that the pencil is "broken" at the point where it enters the water. Snell's law explains this appearance by stating that the paths followed by light rays "bend" every time they intersect a media allowing a different speed of light. Air and water are the media in this example.

The amount of "bend" is determined locally by two angles, θ_1 and θ_2, and two indices of refraction, r_1 and r_2. Snell's law states that the relation

$$r_1 sin(\theta_1) = r_2 sin(\theta_2)$$

must hold at each bend point. (Snell's law is illustrated several times in this thesis. See Figure 55 in Section IV.C for an example where reciprocals of refractive indices are used.) Let B denote the boundary between the two media having different refraction indices and let N denote a normal to B through the point where the ray of light strikes B. θ_1 is the angle between the light ray and N in the medium with index r_1 and θ_2 is the angle between the light ray and N in the other medium.

Reliance on Snell's law is intuitively appealing for the weighted-region problem. Suppose that point P_L is in a low-cost region and P_H is in a high-cost region. The optimal path between the two points must be some perturbation (i.e., warping) of the straight line between them that trades increased distance in the low-cost region for decreased distance in the high-cost region. If B is the boundary between the two regions, the distance tradeoff is achieved by "bending" the path towards a normal to B in the high-cost region and away from the normal in the low-cost region. In Chapter IV, we prove the applicability of Snell's law to the weighted-region problem.

The first requirement for using CDA is that each homogeneous-cost region must be triangularized: each polygon defining a region must be broken up into a set of spatially disjoint triangles. Given the triangularization and a starting point, the CDA applies Dijkstra's algorithm (the uniform-cost strategy) and Snell's law to create a planar subdivision of the representation. The subdivision stores information so that finding the optimal path from the start to any point on a triangle boundary requires little more than indexing the correct answer.

Greatly oversimplifying, CDA uses Snell's law to create disjoint "intervals of optimality" on triangle boundaries that are characterized by "wedges" of minimal-cost paths from the start to that boundary. Snell's law can be used to find the single minimum-cost path within a wedge. This cost is used in place of the (known) node-to-node accrued cost required to execute Dijkstra's algorithm. That is, the cost is used to define minimum-cost wedges (in place of paths) from the start to intervals of optimality (on triangle boundaries) that are progressively more costly (conceptually, farther away) to reach. Relying on the dynamic programming flavor of Dijkstra's algorithm, the algorithm is continued until the minimum-cost wedge for each interval of optimality on every boundary in the problem representation has been found. Thus, at the conclusion of the algorithm, wedges containing the optimal path from the start to every point on all triangle boundaries have been characterized and stored. Given a specific goal, the optimal path can be found by iteratively solving Snell's law within the correct wedge.

The work reported in [Ref. 42] marks a large conceptual advance over other techniques applied to the weighted-region problem and deserves a fuller explanation than has been provided. We avoid discussion in greater detail for two reasons. First, many of the basic principles used in the CDA are also fundamental to the solution presented in Chapters IV and V of this work and there is no need to discuss the same issues more than once. Secondly, the CDA was developed from within the operations research community and is thus primarily oriented towards establishing a firm mathematical foundation. A principal contribution of the work in [Ref. 42] is the establishment of worst-case time and space bounds for the CDA. The algorithm has time complexity $O(n^7L)$ and space complexity $O(n^3)$ where n is the number of boundaries in the triangularization and L relates to precision. To achieve these bounds, the CDA is not constructed for optimal time and space performance in the average case. [Ref. 42] states that some implementation choices were based solely on the need to establish worst-case order classes. The procedure that iteratively solves a given Snell's-law problem is a primary example.

Clearly, establishing a firm mathematical foundation is an important contribution. This having been established, we focus on improving the average-case performance of a Snell's-law-directed solution to the weighted-region problem. There are differences caused by the two approaches. Note that CDA relies on an uninformed strategy, the dynamic programming paradigm as embodied by Dijkstra's algorithm. Chapter V discusses a solution based on an informed strategy, A* search. Although worst-case performance is more difficult to predict, A* search normally performs better than does uniform-cost search (i.e., Dijkstra's algorithm). Recall from Table 4 that A* behavior degenerates to uniform-cost search in the absence of heuristic information (i.e., when the lower-bound evaluation = 0). This example also evinces the importance of heuristics and pruning criteria to the methods described in Chapter V. Thus, for these reasons, we prefer to discuss fundamental issues in an appropriate context.

6. Summary

The general-case weighted-region problem can be solved, although the Snell's-law-based method used in the CDA is the only technique that provides a high degree of solution path accuracy. The wavefront-propagation method relies on a finite, exhaustive graph while Snell's-law-based methods utilize dynamic graphs. The more accurate solution paths generated by the Snell's-law-based method do not suffer from digital bias. The order classes of the Snell's-law-based and lattice-based methods are fundamentally different and the average-case performance of both methods can be improved.

G. SUMMARY

From the previous discussions, we have seen that a solution technique for the general-case weighted-region problem will have several key properties. First, there must be provisions to account for the interaction of subproblems because failure to do so leads to non-optimal solutions such as those provided by the simple localized-improvement model. Specific domain knowledge can been employed to prevent these difficulties as is the case in the VGraph model. Here, knowledge that, in the binary-case, turn points on the optimal path must coincide with obstacle vertices leads to the exhaustive decomposition of the problem into a graph of obstacle vertices which can be intelligently searched. Such decomposition is possible in the binary-case due to the chain of implications:

Straight-Line *implies* Shortest-Distance *implies* Least-Cost

We have seen that the verity of such an implication rests on a uniform-cost being associated with all traversable areas. This uniform-cost premise is not applicable in the general-case weighted-region problem. Thus, the analogous problem decomposition for the non-binary case leads to the imposition of a uniform lattice structure as in the wavefront-propagation technique. Again, an intelligent graph search can be conducted to find an optimal solution. The salient difference is that an unintelligent problem representation limits the accuracy of the search strategy and leads to problems of combinatorial explosion, accumulation of error, and a multiplicity of solutions which appear to represent equal-cost solution paths in the physical environment because they all have the same (digital) cost in the representation. The homogeneous regions approach attempted to establish a more intelligent problem representation by grouping similar points together to form regions. However, this technique also fails to accurately solve the general-case weighted-region problem due to poor representational robustness (not all physical world features can be adequately modeled) and the lack of an appropriate straight-line hypothesis to guide search (moving from region center to region center is inadequate to characterize optimal-cost solution paths). Examination of the binary-case techniques also indicated that a dynamically created graph can lead to greater efficiency by avoiding wasteful computations (when the cost of graph creation cannot be amortized). The Continuous Dijkstra Algorithm combined these last observations, proposing Snell's law as an underlying principle for the general problem. Here Snell's law acts as a local optimality criterion.

The human-like planning systems that we have discussed exhibit several principles. They make use of both domain knowledge and procedural knowledge. In the weighted-region problem, these types of knowledge correspond to topographical knowledge and knowledge of agent capabilities (the agent that will execute the planned solution path). Human-like planning also has an opportunistic element. Although completing multiple tasks at one time is not a component of the weighted-region problem, one can view special cases of terrain features as presenting opportunities for problem decomposition. We have used the example of a bridge as presenting such an opportunity. This concept generalizes to the appearance of a corridor through an otherwise impenetrable obstacle. A simple example is a door in a building. A more important example is the occurrence of a single road through a densely wooded and treacherous mountain area. A suitable solution technique for the weighted-region problem must be able to achieve opportunistic decomposition by recognizing similar situations. Another useful aspect of human-like reasoning is that it is multidirectional. Moreover, directionality is intelligently specified. The wavefront technique is multidirectional, however, omnidirectional search is not an intelligent strategy. Bidirectional search has often been cited as a good strategy due to its limiting properties [Ref. 2,5,16].

The wavefront technique would benefit from bidirectional search in combating combinatorial explosion. The number of nodes examined in this technique is roughly proportional to the area of a circle describing the wave boundary. The area of a circle is πr^2. If waves were propagated from both the start and goal, the sum of their final radii would be approximately equal to the final radius of a single wavefront generated from the start. We know that $a^2 + b^2 < (a + b)^2$ by the amount $2ab$. Thus, combinatorial explosion can be somewhat abated by a simple bidirectional search.

In summary, a suitable solution method for the general-case weighted-region problem could exhibit several properties. These include the use of a basic guiding principle for search (such as Snell's law) that serves as a local optimality criterion, domain knowledge, capability knowledge (knowledge about the abilities of the agent that must execute the planned path), multidirectional (at least bidirectional) and informed search, opportunistic decomposition, and an intelligent problem representation. Also, the solution provided should be in some sense optimal. We note that optimality can be measured by many factors such as time, fuel used, visibility, danger avoidance, and so on. Another consideration is the amount of computation required to obtain the solution. The tradeoff between processing time and optimality must also be considered. We note that humans are able to quickly solve path-planning problems, but not necessarily with optimal results. The graph-theoretic techniques that we have discussed can provide optimal solutions (in terms of the problem representation that they use), but not necessarily quickly. A suitable solution method for the weighted-region problem will achieve the best traits of both.

III. IMPROVING WAVEFRONT-PROPAGATION PERFORMANCE

A. INTRODUCTION

Wavefront propagation is an appealing solution technique for the general-case weighted-region problem because it is conceptually simple, easy to implement and flexible. Also, the method only relies on simple arithmetic operations such as addition and subtraction. Thus, the technique is not greatly affected by numerical errors that can often occur, when using trigonometric functions for example. As a result, the performance of wavefront-propagation algorithms is consistent in most circumstances. However, the simplicity of the algorithm has attendant drawbacks. Increasing the accuracy of wavefront solutions requires increasing representational resolution. We have noted that increasing resolution by a factor of X increases time and space requirements of wavefront propagation by a factor of X^2. This increase is primarily attributable to the uninformed nature of uniform-cost search. The strategy produces optimal solutions (optimal in terms of the lattice-based problem representation) because it is *semi-exhaustive;* it looks everywhere, but only up to a certain point.

Improving the performance of wavefront-propagation algorithms can involve several areas. Preceding sections discussed the difficulties associated with node resolution, link resolution, digital bias and accuracy. All of these problem areas arise from the information loss that occurs when the problem representation is generated. The appropriate cure for these ills lies in the creation of a problem representation that has an information preserving nature, not in devising strategic changes to the search algorithm. Such changes can diminish the impact of representational problems, but control flow is fundamentally the wrong area to address information loss. In this chapter, we do not address the problem-representation issues. Given the representational difficulties, the algorithmic problems involve retrieving the best solution path and decreasing time and space requirements.

A natural question arises: Is it possible to retain the simple nature of wavefront propagation yet overcome the semi-exhaustive character of uniform-cost search? Replacing uniform-cost search by A* search is an effort in this direction. However, recall that exhaustive search can be more effective than informed strategies for tasks that have comparatively low node-generation costs. In the problem representation used by wavefront propagation, the cost of node generation is low.

In this chapter, we examine the operation of the wavefront-propagation algorithm in greater detail. Our effort is directed towards improving the performance of the algorithm so that we can establish a baseline standard of performance for weighted-region problem solution techniques. In Chapter VI we compare the performance of a Snell's-law-based solution technique against this standard. Four new versions of wavefront propagation are introduced. These are named the bidirectional strategy, the heuristic-selection strategy, the ellipse strategy and the ellipse-and-heuristic-selection strategy. The performance of these strategies are compared to known wavefront-propagation algorithms (unidirectional, A*-based, and A_ϵ*-based). We first address methods of retrieving solutions once the goal has been found. Then, simple strategic alterations that decrease the time requirements of the algorithm are introduced. These modification have low overhead yet improve average-case performance. Finally, a performance comparison of different strategies is provided and the results of the comparison are summarized.

B. DEFINING THE PROPAGATION PROCEDURE

There are two principal methods of retrieving a solution path once the goal has been found. The first is to save *snapshots* of the wavefront as it progresses towards the goal. Saving a snapshot of the wavefront requires saving the exact location of the entire wavefront at a specific instance of time. Once the goal is reached, a gradient-tracing routine can project normals from the goal, through each snapshot, back to the start, determining a solution path. This method has three primary deficiencies. First, gradient tracing invites resolution problems that affect the algorithm as well as the problem representation. Deciding how many snapshots should be saved and at what interval is arbitrary. Secondly, computing the intersections between normals and wavefront snapshots can increase time requirements (a factor we wish to decrease) and, again, the amount of increase is a factor of resolution. Note that, for each snapshot saved, the first intersection of the normal with the snapshot must be found. (There will be two such intersections. one on each side of the snapshot.) Even though only one path must be found by gradient tracing, we should to avoid this post-processing step if a less time consuming method is available. Finally. gradient tracing is not the simplest method of recovering a solution from a uniform-cost search. The solution retrieval question is simply put: given a node on a path, where is the parent of that node? This information is readily available during the search. Thus, when ancestry information (i.e., information that specifies the parent of each node) is preserved during the search, the solution retrieval question is answered by tracing the ancestry of the goal node. The choice between maintaining ancestry records or saving snapshots for a gradient-tracing routine involves the classic time/space tradeoff. Keeping ancestry records requires storage, but the time required to retrieve a solution is decreased. Saving snapshots also requires storage, the amount of which is determined by the resolution. If every second wavefront is saved, approximately one half of the nodes expanded must be stored as different snapshots.

Preserving ancestry records amounts to maintaining *backpointers* during node expansion. When a child is generated, a backpointer from the child to the parent must be set. (We only allow one parent for each node as discussed below.) Given a static, eight-way connected graph, the minimal storage required to save backpointers is 3 bits per generated node. To see this, note that in an eight-way connected graph, the parent of any node must be one of the node's 8 neighbors. Thus, storing one of eight directions suffices to specify a link to the parent for any node and choosing one of 8 alternatives requires only three bits of information. This is a minimal storage requirement. If storage is not a limiting factor. preserving an unencoded specification of the parent is more convenient. Specifically, storing the Cartesian coordinates of the parent or an index to an array that contains the parent facilitates tracing backpointer links.

Given that backpointers are maintained, there are two times when these can be set. A pointer may be set as soon as a node is generated or setting the pointer can be delayed until the node is eligible to be put on the wavefront. The specific knowledge of the wavefront-propagation operation necessary to understand this issue is developed below.

Because there is a predetermined finite number of links (we assume 8 in the following discussions) associated with each node, there is no need to explicitly store any link. Instead, we can use the indices of a two-dimensional array to provide this information. As an example, suppose that the coordinates of a node are (X, Y). Then, the eight neighbors of this node have coordinates (clockwise from the northern neighbor) $(X, Y+1)$, $(X+1, Y+1)$, $(X+1, Y)$, $(X+1, Y-1)$, $(X, Y-1)$, $(X-1, Y-1)$, $(X-1, Y)$ and $(X-1, Y+1)$. When links are implicit, the cost of traversing a link must be associated with the node itself, just as is done in the ADS dynamic programming model [Ref. 36]. Conceptually, nodes are cells that have static *cost rates;* the cost associated with passing through the node (from any other connected node). Note that this organization means that every link associated with a given node has identical link cost.

One way to view the operation of the wavefront-propagation algorithm is as a simulation. Suppose that the minimal-time path is desired. Then, the wavefront simulates all possible locations of an agent at successive instances of time. At time zero, the agent is at the start. After one time unit passes, the agent can be in any one of eight possible locations that are all one cost unit distant from the start. Thus, at time zero, only the start node is *on the wavefront*. After one time unit passes, up to eight nodes are on the wavefront.

Determining whether or not a node can be placed on the wavefront depends on the cost to traverse the node and the direction of travel through the node. If the cost of traversing through a node is C, then at least C time units must pass before the node can be placed on the wavefront. The direction of travel through the node is important because of the representation. Each orthogonal neighbor of a node is one time unit distant from that node. However, each diagonal neighbor is $\sqrt{2}$ units distant. Suppose that the simulation is at time zero and that each neighbor of the start node has unit cost. Then, at time 1, each orthogonal neighbor of the start is reached and can be placed on the wavefront. However, only $1/\sqrt{2}$ of the distance from the start to each diagonal neighbor can be traversed in one time unit so that none of these neighbors are reached. Using a factor of $1/\sqrt{2}$ for the diagonal links means that, in some instances, the propagation effect can *overflow* a diagonal neighbor and continue into a node that is not an immediate neighbor of the node being expanded. That is, the wavefront can pass entirely through a neighboring node and move on to the neighbor's neighbor. Also note that in the explanation we have provided so far, the factors 1 and $1/\sqrt{2}$ are tied to allowing only one time unit to pass between each computation of the wavefront's progress. Incrementing time at a rate of $\sqrt{2}$ instead of 1 associates a factor of $\sqrt{2}$ with orthogonal neighbors and 1 with diagonal neighbors. The larger time interval is desirable because it allows the wavefront to "take longer steps" towards the goal.

To implement this simulation, a cost rate must be associated with each node. Then, for each node on the wavefront, inspect the node's neighbors (i.e., expand the node, generating its children) to see if they are eligible for addition to the wavefront. A node is eligible if it is not in an obstacle area, it has not already been put on the wavefront, and the wavefront could pass through that node during the current propagation increment of the wavefront location. Specifically, for each orthogonal neighbor, retrieve the cost to traverse through that neighbor and decrement it by $\sqrt{2}$. If the decremented cost is equal to zero, the neighbor can be added to the wavefront. If the decremented cost is less than zero, the neighbor can be added to the wavefront and the decremented cost (equal to the negative of the ovorflow amount) must be propagated through the neighbor until it reaches zero. If the decremented cost is greater than zero, the neighbor cannot be added to the wavefront. However, the fact that some progress has been made towards reaching the neighbor must be saved. This is achieved by altering the stored cost associated with traversing through the neighbor. (Note that to solve a new problem, the original cost for each node must be restored.) The same procedure is repeated for eligible diagonal neighbors except that stored cost rates are decremented by 1 instead of by $\sqrt{2}$. Also, assuming integral cost rates, overflow is not an issue for diagonal neighbors. Each explored node is removed from the wavefront when all of its neighbors have either been placed on the wavefront or been declared ineligible for expansion.

Table 6 provides a procedural definition of expanding a node on the wavefront. The definition assumes that the cost to traverse through a node is stored in a two-dimensional array *Cost* so that if X and Y are the Cartesian coordinates of a node, then $Cost(X,Y)$ yields the cost rate for that node. Also, if $Cost(i,j)$ is less than zero, the node at coordinates (i,j) is in an obstacle region and is not eligible to join the wavefront. The definition also depends on an

TABLE 6
EXPANDING WAVEFRONT NODES

```
Expand(X,Y)
  {
  Set Neighborcount = 0
  For each of the 8 neighbors of node (X,Y)
    {
    Generate the neighbor's coordinates (Xn,Yn)
    if Cost(Xn,Yn) > 0
      {
      if (Xn,Yn) is an orthogonal neighbor
        Newcost = Cost(Xn,Yn) - $\sqrt{2}$
      otherwise
        Newcost = Cost(Xn,Yn) - 1
      if Newcost <= 0
        {
        Neighborcount = Neighborcount + 1
        Add (Xn,Yn) to the wavefront
        Set Cost(Xn,Yn) = 0
        }
      if Newcost < 0
        Overflow(Xn,Yn,X,Y,Newcost)
      if Newcost > 0
        Cost(Xn,Yn) = Newcost
      }
    otherwise
      Neighborcount = Neighborcount + 1
    }
  If Neighborcount = 8
    Delete node (Xn,Yn) from the wavefront
  }
```

overflow procedure to continue the propagation of the wavefront when required. This procedure is strictly defined below. The central idea of overflow can best be explained by using the passage of time simulation view of wavefront propagation. The wavefront overflows through a cell when enough total time has passed so that the wavefront can cover the entire distance through the cell and make progress into a neighboring cell during the same time interval.

The procedure of Table 6 does not include any provision to set backpointers for newly-generated nodes. This is an important consideration for those neighbors that cannot be added to the wavefront due to high cost rates. For these nodes, the entry in the *Cost* array is updated. However, the updated entry reflects the progress made towards reaching that node from a specific parent. No other potential-parent node can be allowed to reference the same, updated cost rate. We store the parent of each node in an array $Parent(X,Y)$ so that a reference to $Parent(X,Y)$ yields the backpointer to the parent of the node at coordinates X,Y.

The issues involved in choosing to set backpointers as soon as possible or as late as possible should now be apparent. The earliest that a parent can be chosen is when a node has been generated and declared eligible for expansion. Setting the pointer at this time means that no other potential parent can be allowed to generate this node as an eligible neighbor. Setting backpointers late requires maintaining an updated cost for each potential parent. Then, once all the potential parents of a node have been explored, the parent that allows the wavefront to make the greatest amount of progress through (or to) the node can be chosen as the permanent parent. Once the permanent parent is selected, the backpointer can be set and the node is declared ineligible for expansion from any other potential parent.

Setting backpointers as late as possible requires more time and space to realize a very localized improvement. Figures 7 and 8 clearly show the difference between the two methods. (We note that this analysis applies to propagating the wavefront through uniform-cost areas.) Both figures depict backpointer trails from all nodes explored during the search back to the start node (at the center).

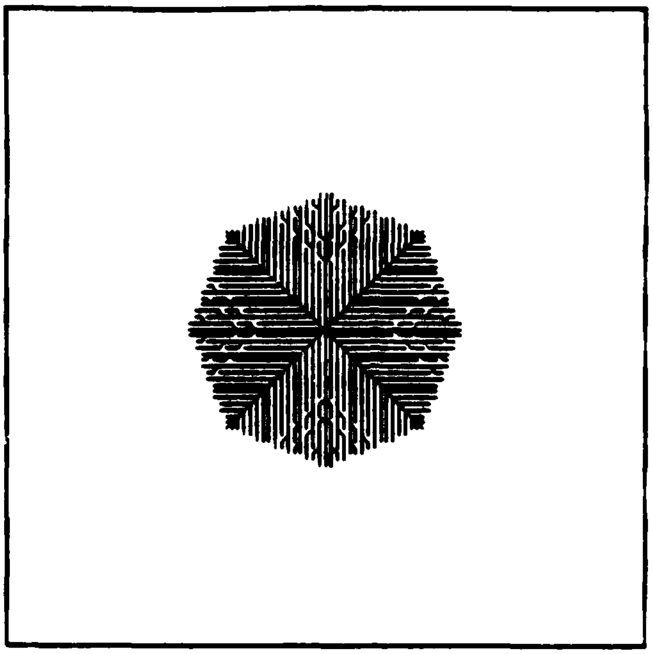


Figure 7. Paths Derived From Setting Backpointers As Soon As Possible

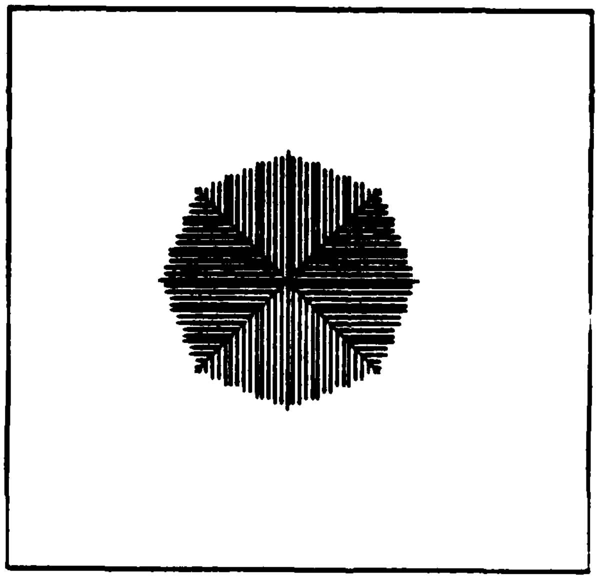


Figure 8. Paths Derived From Setting Backpointers As Late As Possible

Both wavefronts were propagated through uniform-cost-rate nodes (and thus, no obstacles are involved). The wavefronts were propagated for 15 time intervals.

expanding 372 nodes. Figure 7 reflects the "as soon as possible" choice while Figure 8 depicts the linkage pattern produced by the "as late as possible" strategy. The latter strategy required 2.8 times as much time and 3.7 times as much storage to effect a very localized change in the ancestry records. The former less expensive strategy favors paths having more turn points, thus making better "stair-step" approximations to straight lines. Also, wavefront propagation is inherently a satisficing strategy, due to the problem representation. The amount of solution improvement offered by the more expensive strategy does not seem justified. For these reasons, setting backpointers as soon as possible is the best alternative. Table 7 provides a new procedural definition of wavefront node expansion that maintains ancestry information. We assume the existence of an array, *Parent*(X, Y), that stores the coordinates of the parent to the node located at coordinates (X, Y).

The definitions of Tables 6 and 7 both rely on a procedure that controls overflow situations. Overflowing through a node is similar to expanding a node. However, there is a directional aspect involved in overflow cases that does not directly affect normal node expansion. The overflow stems from a specific parent and "flows" in a specific direction. Altering the direction (from the parent) also changes the amount of overflow. Thus, the problem is similar to the problem of setting backpointers. In an overflow situation, the propagation must be continued in the same direction. Thus, the input parameters to the overflow procedure (as in Tables 6 and 7) include the node that overflow propagates through, the parent of that node, and the amount of overflow. Table 8 provides a definition of the overflow procedure. Note that using a factor of $\sqrt{2}$ for orthogonal neighbors and 1

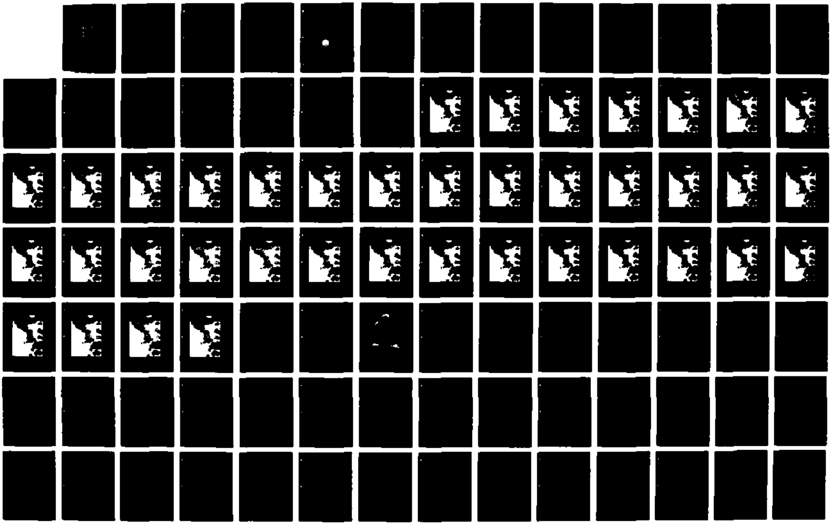
TABLE 7
EXPANDING WAVEFRONT NODES & SETTING POINTERS

```
Expand(X,Y)
  {
  Set Neighborcount = 0
  For each of the 8 neighbors of node (X,Y)
    { Generate the coordinates of the neighbor, (Xn,Yn)
    if Parent(Xn,Yn) is undefined
      Set Parent(Xn,Yn) = (X,Y)
    if((Cost(Xn,Yn) > 0) and (Parent(Xn,Yn) = (X,Y))
      { if (Xn,Yn) is an orthogonal neighbor
        Newcost = Cost(Xn,Yn) - $\sqrt{2}$
      otherwise
        Newcost = Cost(Xn,Yn) - 1
      if Newcost <= 0
        {Neighborcount = Neighborcount + 1
         Add (Xn,Yn) to the wavefront
         Set Cost(Xn,Yn) = 0
        }
      if Newcost < 0
        Overflow(Xn,Yn,X,Y,Newcost)
      if Newcost > 0
        Cost(Xn,Yn) = Newcost
      }
    otherwise
      Neighborcount = Neighborcount + 1
    }
  if Neighborcount = 8
    Delete node (X,Y) from the wavefront
  }
```

for diagonal neighbors, and relying on integral cost rates, means that overflow never occurs through diagonal neighbors. Also, the overflow procedure cannot remove a node from the wavefront. By definition, a node through which overflow propagates has not yet been expanded and, thus, cannot be on the wavefront.

There is one other improvement that can be made to wavefront propagation as defined by Table 7. It is not necessary to inspect all eight neighbors for the possibility of being reached by the wavefront as it propagates

AD-A183 900 SOLVING A CLASS OF SPATIAL REASONING PROBLEMS: 2/5
MINIMAL-COST PATH PLANNING IN THE CARTESIAN PLANE(U)
NAVAL POSTGRADUATE SCHOOL MONTEREY CA R F RICHBOURG
UNCLASSIFIED JUN 87 F/G 12/9 NL



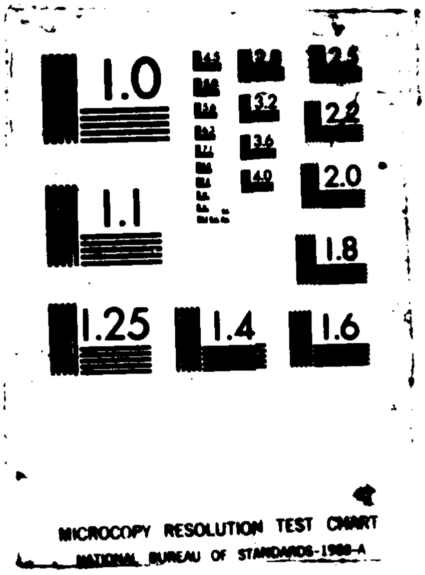

1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

TABLE 8
PROPAGATING OVERFLOW

```
Overflow(Xn,Yn,X,Y,Amount)
   {
   Determine the direction of overflow by setting
        Dx = Xn - X, Dy = Yn - Y
   Find the node, (Xp,Yp), overflow propagates to
        by Xp = Xn + Dx, Yp = Yn + Dy
   if Parent(Xp,Yp) is undefined
      Set Parent(Xp,Yp) = (Xn,Yn)
   if((Cost(Xp,Yp) > 0) and (Parent(Xp,Yp) = (Xn,Yn))
      {
      Newcost = Cost(Xp,Yp) + Amount
      if Newcost <= 0
         {
         Add (Xp,Yp) to the wavefront
         Set Cost(Xp,Yp) = 0
         }
      if Newcost < 0
         Overflow(Xp,Yp,Xn,Yn,Newcost)
      if Newcost > 0
         Cost(Xp,Yp) = Newcost
      }
   }
```

node. If (X,Y) is the node being expanded, then only those neighbors of (X,Y) that have an undefined parent (all nodes that have not been reached by the wavefront have undefined parents) or already have (X,Y) as a parent need be inspected. Clearly, the wavefront cannot be propagated back to the parent of (X,Y). Reasoning about the direction of wavefront flow to reach (X,Y) also eliminates other nodes from consideration. Figure 9 depicts a situation where the wavefront has been propagated from node P to node N and N is currently under expansion (for illustrative purposes, Figure 9 adopts the grid of cells view of the problem representation). For illustrative purposes, the eight neighbors of N are labeled 1 through 7 and P. Generally, the wavefront should have reached node 1 and node P at the same time. If not, node 1 and node N must both have node P as a parent. The same situation is true regarding node 7. Also, nodes 2 and 6 must have been claimed as children by node P at the same time that the wavefront reached node N. Thus, nodes 1, 2, 6, 7 and P must already have parent nodes other than node N. This holds, regardless of node cost, since the "as soon as possible" scheme is in use. Therefore, only nodes 3, 4 and 5 need be inspected for further propagation from node N.

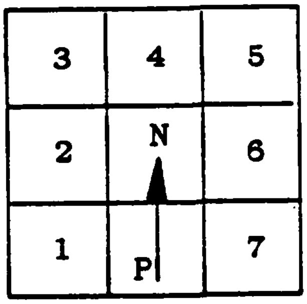


Figure 9. Propagation to Neighboring Nodes

The direction of wavefront travel (from P to N in Figure 9) is important when determining those three neighbors that can be reached by further propagation of the wavefront. As there are eight possible directions of approach to a node, there are eight sets of neighbors that can be reached from that node. Figure 10 depicts each possible case. The arrows in Figure 10 denote the direction of propagation to node N, the node under expansion. The eligible neighbors of N are enclosed by a dark border.

Figure 11 shows the linkage pattern resulting from a wavefront propagation when only three neighbors are inspected for possible expansion. The problem is

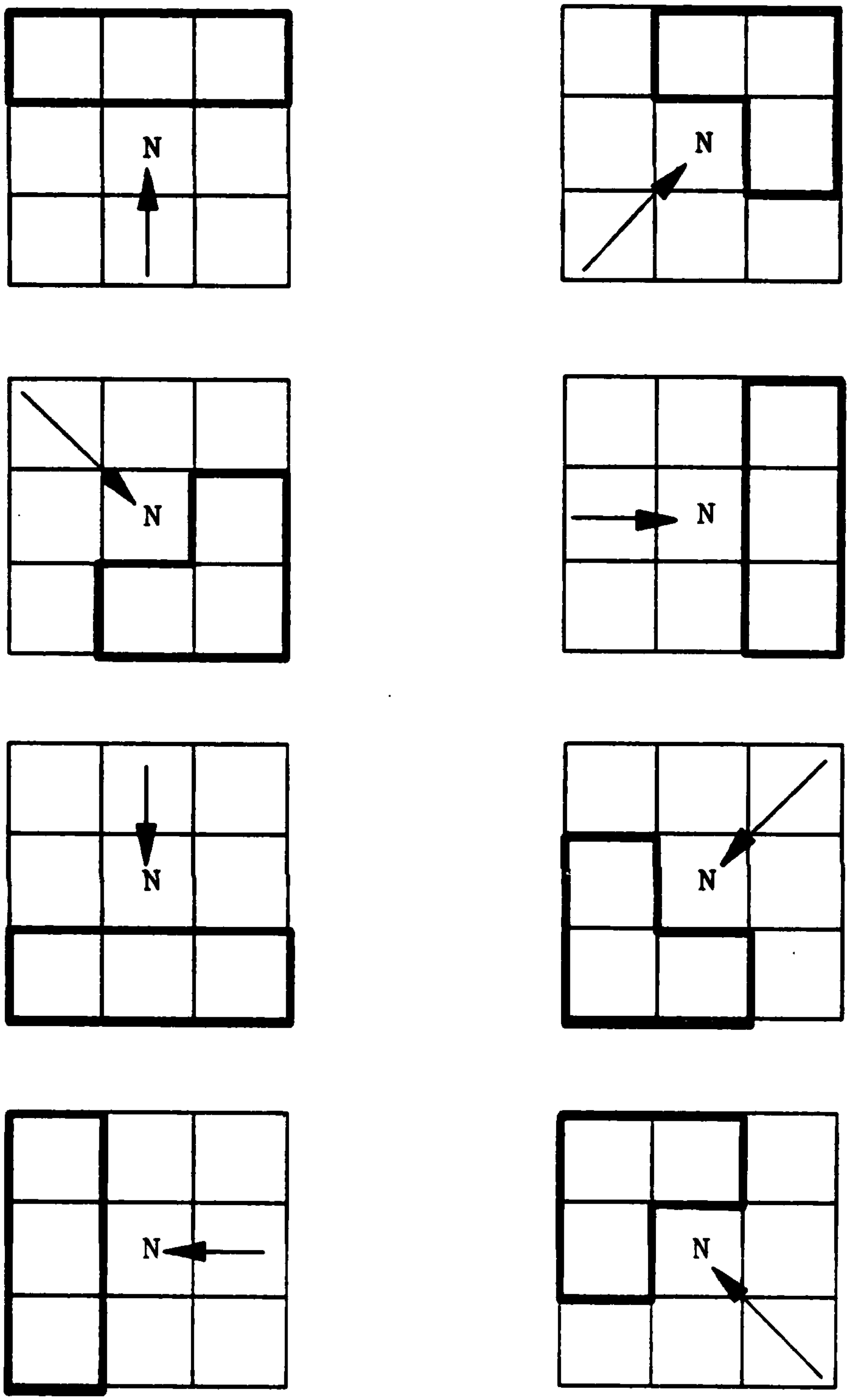


Figure 10. Eligible Neighbors

the same as was used to create Figures 7 and 8. Note that Figure 7 and Figure 11 are identical. Both strategies were executed by a C-Prolog interpreter running on an Integrated Solutions Optimum V workstation under Berkeley UNIX, System 4.2. The three-neighbor wavefront-propagation search required approximately 25% less time to complete than the eight-neighbor strategy used to create Figure 7. C-Prolog does not support array data structures. List structures are normally used in their place and searching through a list is more time consuming than directly accessing an array element. Table 9 presents a procedural definition of expanding a node while inspecting only three neighbors. This definition substitutes one array reference for the inspection of five neighbors (for eligibility to join the wavefront) when compared to the procedural definition in Table 7.

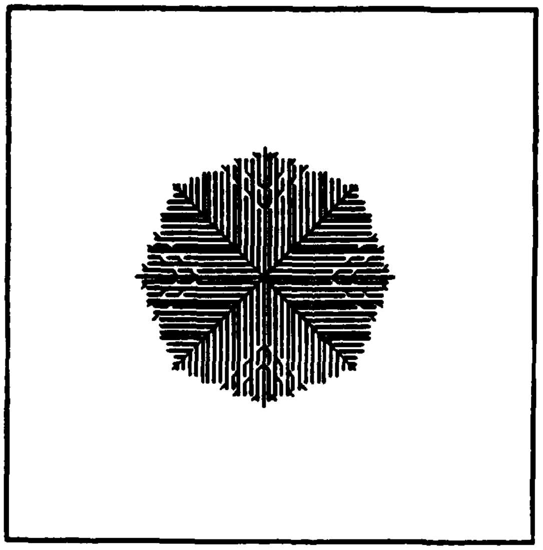


Figure 11. Three-Neighbor Wavefront Backpointer Pattern

In C-Prolog, retrieving the ancestry information is more expensive (in time) than it would be in most languages supporting arrays. Thus, the 25% reduction in execution time is conservative. Note that the definition of Table 12 requires two

TABLE 9
3 NEIGHBOR NODE EXPANSION

```
Expand(X,Y)
   {Set (Xp,Yp) = Parent(X,Y), Dx = X - Xp, Dy = Y - Yp
   if Dx = 0
      {F1 = Sub-expand(X-1,Y+Dy,X,Y,1)
       F2 = Sub-expand(X,Y+Dy,X,Y,$\sqrt{2}$)
       F3 = Sub-expand(X+1,Y+Dy,X,Y,1)
      }
   else if Dy = 0
      {F1 = Sub-expand(X+Dx,Y-1,X,Y,1)
       F2 = Sub-expand(X+Dx,Y,X,Y,$\sqrt{2}$)
       F3 = Sub-expand(X+Dx,Y+1,X,Y,1)
      }
   else
      {F1 = Sub-expand(X,Y+Dy,X,Y,$\sqrt{2}$)
       F2 = Sub-expand(X+Dx,Y+Dy,X,Y,1)
       F3 = Sub-expand(X+Dx,Y,X,Y,$\sqrt{2}$)
      }
   if (F1 + F2 + F3) = 3
      Delete (X,Y) from the wavefront
   }

Sub-expand(X,Y,Xp,Yp,Amount)
   {if Parent(X,Y) is undefined
      Set Parent(X,Y) = (Xp,Yp)
    if((Parent(X,Y) = (Xp,Yp) and (Cost(X,Y) > 0))
      {Newcost = Cost(X,Y) - Amount
       if Newcost <= 0
          {Add (X,Y) to the wavefront
           Set Cost(X.Y) = 0
           Returnvalue = 1
          }
       if Newcost < 0
          Overflow(X,Y,Xp,Yp,Newcost)
       if Newcost > 0
          {Cost(X.Y) = Newcost
           Returnvalue = 0
          }
      }
    else Returnvalue = 1
    Return(Returnvalue)
    }
```

procedures. Procedure *Expand* determines the eligible neighbors. Procedure *Sub−expand* propagates the wavefront, if possible, and returns a value so that *Expand* can remove the node from the wavefront when necessary.

C. DECREASING WAVEFRONT-PROPAGATION TIME REQUIREMENTS

In this section, we introduce two new concepts that can be used to decrease the time required by wavefront-propagation algorithms to arrive at a solution path. First, we show how bidirectional search can be used in this algorithm. Secondly, we introduce the notion of a global bound which limits the portion of the lattice that must be searched.

1. Bidirectional Strategies

In Section II.G we noted that a bidirectional strategy has the potential to decrease the number of nodes expanded during wavefront propagation. This analysis was based on assuming a circular shape for the wavefront that occurs when it propagates through uniform-cost areas. (We note that circularity is not required; it simply makes the analysis less complicated.) When the assumption holds, the wavefront at solution approximates a circle of radius r where r is the cost of the path from start to goal. Suppose that, instead of propagating one wavefront from the start, two wavefronts are propagated, one from the start and one from the goal. At solution, the two circles have radii r_S and r_G and $r_S + r_G = r$. The number of nodes expanded is approximated by the area of the circle. Clearly, the sum of the areas of the two smaller wavefronts is less than the area of the single, larger wavefront.

There is some overhead associated with bidirectional wavefront propagation and, if the objective of exploiting bidirectionality is to reduce time, that overhead should be small. The expansion procedure defined in Table 9 is low-level. In the definition, we have not provided a way for the procedure to know when the goal has been found. Thus, there must be some higher-level routine that selects nodes on the wavefront for expansion and determines when a solution has been reached. In standard unidirectional wavefront propagation, a solution is available when the goal node is reached through a neighboring node that is on the wavefront. Thus, whenever an eligible neighbor is generated, its X and Y coordinates must be compared to those of the goal. If the coordinates match, the goal has been found and the low-level procedures (such as Expand and Sub-expand in Table 9) can set a notification flag. In total, detecting a solution requires three comparisons for each expanded node; one against a flag value and one each for the X and Y coordinates of the goal.

This simple termination criterion does not work when using a bidirectional strategy. Instead, a solution is available when the two wavefronts touch. However, using some of the structures already available, determining wavefront intersection is also an easy task. Suppose that we initialize the *Cost* array entry for the start node to be 0 and for the goal node to be -1. Then, each time a node is added to the wavefront (when its cost is ≤ 0), we set the *Cost* array entry for that node to be equal to the entry for its parent node (instead of an arbitrary zero cost). The two wavefronts touch when an ineligible neighbor node has a *Cost* array entry different from that of the node under expansion. A slight complication arises in that we have already assumed that unreachable nodes (i.e., nodes inside obstacle areas) are identified by negative costs. Assume these nodes all have costs of -2. Then, if a node ineligible for expansion has a *Cost* array entry greater than -2 and different from the entry for the node under expansion, a flag can be set signaling that a solution is available. Again, three comparisons are required to detect a solution. However, two comparisons involve inequality (i.e., greater than and not equal) and there is an added level of indirection since array entries must be compared. Thus, there is a slight machine-dependent increase in overhead to conduct a bidirectional search. The performance of bidirectional and unidirectional strategies are compared in Section III.D.

Using a bidirectional strategy also allows some flexibility. It is not necessary to expand both wavefronts uniformly. The presence of obstacles tends to decrease the time requirements of wavefront propagation because fewer nodes are eligible to join the wavefront, keeping the size of the wavefront small (relatively). Propagating a wavefront out of a "box canyon" defined by obstacles is less expensive than propagating the wavefront in 360 degrees. Thus, when using a bidirectional strategy, the algorithm can take advantage of this fact and select the smaller wavefront to expand during each time cycle. Again, there is some overhead in determining the smaller of the two wavefronts. If the wavefronts are maintained in separate one-dimensional arrays, simply comparing the indices of the last used array positions provides the relative size of the two wavefronts. Thus, the time overhead is low. This strategy is referred to as the heuristic-selection method in the performance comparisons of Section III.D.

Using more than two separate wavefronts does not seem to be a viable option. For each path-planning problem, there are two points that are known. a priori, to be on the optimal-cost solution path. These points are the start and goal. A characteristic of the weighted-region problem that makes it difficult to solve is that it is not readily decomposable. Intermediate points that must also be on the solution path are not apparent in most cases. Since propagating wavefronts from points not on the optimal solution path is wasted effort, using more than two wavefronts is not helpful.

A final comment on bidirectional wavefront propagation relates to the maximum error in the cost of a solution path. Recall that the maximum (cost) error in a solution derived from a unidirectional strategy is 8% (see Figure 5) and that this error occurs when the goal is a midpoint of a chord approximating a 22.5 degree arc. In bidirectional wavefront propagation, the maximum error occurs when the two wavefronts touch at midpoints of chords, both approximating 22.5 degree arcs. This situation can arise when the physical relationship of the start and goal is similar to that depicted in Figure 12 (where the start is labeled s and the goal g). Let $t=r_1+r_2$ (as in Figure 12) be the true cost of the s-to-g path. From previous discussions (Section II.2.E.a) we know that the chords touch when $\delta_1=1.08d_1$, $\delta_2=1.08d_2$, $\rho_1=1.08r_1$ and $\rho_2=1.08r_2$, as in Figures 12 and 13. In Figure 13, a solution having cost $\rho_1+\rho_2=1.08r_1+1.08r_2=1.08t$ is reported. Thus. the maximum (cost) error in the bidirectional strategy is. again. approximately 8%. Therefore, the maximum error in the cost of a solution path is not increased by using bidirectional search.

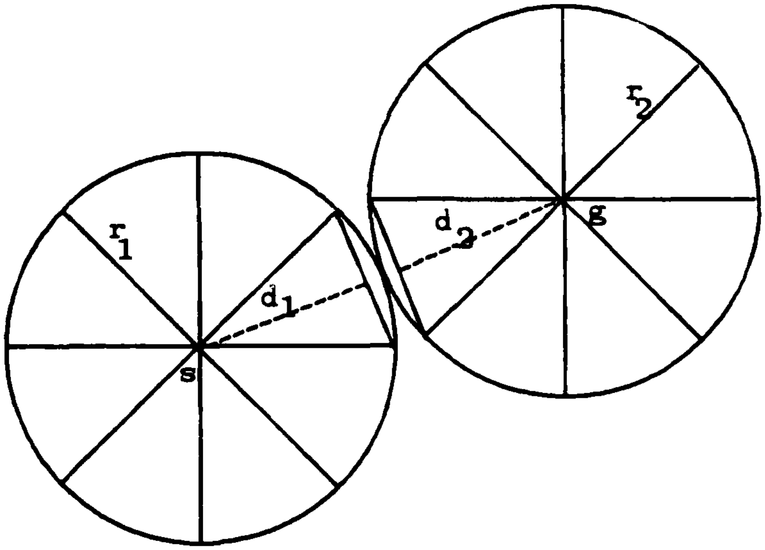


Figure 12. A Solution Should Be Reported When "Perfect" Wavefronts Touch

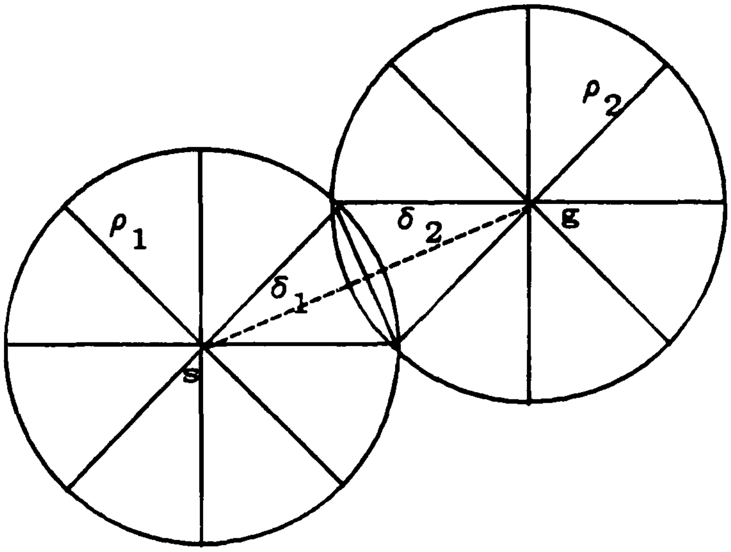


Figure 13. A Solution Is Reported When Approximating Chords Touch

2. Physical Bounds

The semi-exhaustive nature of uniform cost search has been noted. The procedure attempts to examine every neighboring node within a 360 degree arc about the start. Clearly, those nodes neighboring the start that lead away from the goal are less likely to be on the solution path than those nodes that are close to a straight start-to-goal line segment. The heuristics employed by informed strategies attempt to recognize the likelihood that an arbitrary node may be on the solution path. As an example, the Euclidean distance between two points (nodes) can be used to provide a good lower-bound estimate on the cost of a path between those two points. However, measuring Euclidean distance requires an expensive square root function. Also, informed strategies use ordered data structures (possibly linked lists stored as arrays such that each array entry contains a data element and a pointer to the next data element), introducing more overhead. The costs of using evaluation functions and maintaining ordered lists must be paid each time a node is added to the wavefront. When resolution is high, at the pixel level for example, overhead costs can mount quickly.

A one-time overhead heuristic is achieved by physically bounding the search space before the search process begins. Suppose that there is a feasible solution path (a start-to-goal path that stays out of obstacle areas) to the weighted-region problem. Let the cost of the feasible solution path be C_p. The optimal-cost solution path must, by definition, have cost less than or equal to C_p. Also, there must be some optimal cost rate, C_O, associated with the problem representation. Given C_p and C_O there must be a distance D_b such that a path covering D_b at cost C_O has cost equal to C_p, $D_b = C_p / C_O$. Since the optimal-cost solution path must have cost less than or equal to C_p, it must travel a distance less than or equal to D_b. Thus, D_b is a bounding distance. An ellipse that has the start and goal as foci and constructed such that, for each point on the ellipse boundary, the distance from the start to that point plus the distance from the goal to that point is equal to D_b, must contain all start-to-goal paths having distance less than D_b. Thus, the coordinates of the ellipse boundary form physical limits on the location of any part of the optimal-cost solution path. (Note that this is a slightly different version of the idea used in the branch-and-bound search strategy.)

Wavefront propagation can make use of such a physical bound by considering the ellipse boundary as an obstacle. Using this convention. the wavefront is never allowed to propagate outside of the ellipse. Also, there is no additional overhead incurred during the search since there is already a requirement that each node be inspected for eligibility. All overhead is incurred as a one-time cost, before the search begins. A binary-case algorithm, even simple localized-improvement, can ignore the cost rates for passable areas and find a feasible solution on which to base ellipse construction. The comparisons of Section III.D include data for a strategy based on bidirectional search within a limiting ellipse (which we denote ellipse).

D. PERFORMANCE COMPARISONS

This section presents the results achieved by different variations of the wavefront-propagation technique when applied to the same problems. The area-cost map used for testing represents terrain at Point Lobos, California. The map features a ternary terrain classification scheme, i.e., each point in the environment is either impassable, traversable at high cost or traversable at low (optimal) cost. The cost-rate ratio of high-cost traversable areas to low-cost traversable areas is 2:1. The area-cost map was designed to be appropriate for the Adaptive Suspension Vehicle, constructed at The Ohio State University [Ref. 44]. The actual terrain was physically inspected in order to manually assign cost rates to regions on the area-cost map that would reflect the capabilities of this vehicle.

In this section, we present the time required and nodes (pixels) explored by each of six different wavefront strategies to solve the same problems. The first four methods, unidirectional, bidirectional, heuristic-selection and the ellipse (bidirectional without using heuristics) method have been discussed. Wavefront strategies relying on the A^* and A_ϵ^* algorithms are also included in the comparison. The A^* and A_ϵ^* variants both rely on a heap data structure to maintain the ordered Open list. Both strategies use Euclidean distance at optimal cost as the lower-bound evaluation (the h(n) function). All routines run in compiled C on a multiuser, IRIS 2400 workstation under UNIX System V. The time results do include some CPU time dedicated to IRIS graphics tasks. However, the graphics overhead is approximately the same for each method and if any bias is present, those strategies expanding fewer nodes are favored. Thus, the time measures can only be considered as indicative of relative performance. The time performance cited for the ellipse method does not include the time required to achieve an initial feasible solution (as this portion of the strategy was accomplished manually and provided to the test algorithm). Here, initial solutions are simple binary-case solutions where the cost of traversing regions is ignored.

Thus, the shortest-distance paths that do not intersect obstacle areas are used as initial solutions. (Feasible binary-case solutions are not difficult to generate and thus do not require much computation time.) Thus, while the timing marks for the ellipse method are not totally accurate, they are indicative of the method's relative performance.

Figures 14 through 43 depict the results obtained by each strategy. (Note that these figures are all placed at the rear of this chapter.) In these figures, the darkly shaded polygons represent obstacle areas. Lightly shaded polygons depict high-cost traversable areas and the unshaded background area is the low-cost traversable area. The figures show the location of the wavefront(s) at solution and the solution path. The solution, start, goal and wavefront(s) are usually labeled. Some labels are omitted for clarity of individual figures. For bidirectional strategies, the wavefront centered at the start is labeled *s wavefront;* the wavefront emanating from the goal, *g wavefront*. The figures reflecting ellipse-based strategies also show the limiting ellipse as a heavy line. The solution path is a heavy line between two circles, each of which contains either the start or goal. Each node remaining on the wavefront at solution is shown as a single darkened pixel. The pixels form line segments describing the entire wavefront(s), which may be disconnected. The disconnected portions arise when the wavefront cannot be propagated through some area, an obstacle area for example.

Figures 14 through 19 depict solutions to the first problem, denoted Problem A. Figures 15 and 16 are very similar, reflecting the inability of the heuristic-selection method to improve performance on this problem. This is because the high-cost region near the start and the edge of the map near the goal tend to keep both the s and g wavefronts expanding at close to the same rate. Also, note the search pattern of A_ϵ* depicted in Figure 19. This strategy allows some nodes to be skipped over so that the wavefront is not contiguous. Some unexplored nodes remain in the interior of the wavefront. This is a general behavior pattern for the A_ϵ* algorithm that is reflected in several figures. Figures 20 through 25 depict solutions to the second problem, Problem B. Again, performance is affected by the edge of the map. Also note that the unidirectional strategy provides a different solution path (near the goal), an effect of digital bias.

Figures 26 through 31 depict a problem where the shortest-distance path is the optimal-cost path. The width of the high-cost region intersected by the solution path is small enough that the region becomes inconsequential. In this problem, the start is located in a "box canyon" and the heuristic-selection method does affect performance. Also note that the remaining wavefront for the A_ϵ* strategy is so small that the solution path completely hides it from view. However. the A_ϵ* method also yields the least-accurate solution.

Problem D solutions are depicted in Figures 32 through 37. Again, note the inability of the heuristic-selection method to improve performance. Also. there is a large difference between the search patterns produced by A_ϵ* in Problems C and D. Traveling longer distances through high-cost regions confuses this strategy.

Figures 38 through 43 present solutions to Problem E. Note the great increase in the area covered by unidirectional search, due to the higher-cost solution path. Also, the heuristic-selection method has a great effect on this problem. It produces a very different search pattern from the simple bidirectional strategies.

TABLE 10
PERFORMANCE COMPARISON

Strategy	Problem Number	Figure Number	CPU Time (seconds)	Nodes Expanded	Order of Performance
Uni-directional	A	14	37.65	40585	6
	B	20	39.65	46833	5
	C	26	5.46	7564	5
	D	32	23.13	29668	5
	E	38	65.95	75149	5
Bi-directional	A	15	23.23	25946	4
	B	21	20.15	25345	3
	C	27	7.57	10363	6
	D	33	18.43	22198	3
	E	39	37.05	44429	4
Heuristic-Selection	A	16	22.73	25702	3
	B	22	17.38	22575	1
	C	28	4.70	6612	4
	D	34	17.95	21741	2
	E	40	28.62	35172	2
Ellipse	A	17	13.80	14546	1
	B	23	18.18	23117	2
	C	29	2.22	2964	2
	D	35	5.68	7105	1
	E	41	14.30	15335	1
A*	A	18	19.50	4736	2
	B	24	36.83	8609	4
	C	30	3.60	1024	3
	D	36	19.70	4604	4
	E	42	30.05	7155	3
A_ϵ*	A	19	27.71	3042	5
	B	25	84.18	4623	6
	C	31	1.07	360	1
	D	37	80.42	3813	6
	E	43	89.28	2935	6

The exact time and space performance of each strategy on each problem is presented in Table 10. The table contains a column labeled "Order of Performance" that rank orders each strategy, 1 through 6, by time required to solve each problem. In Table 11, the mean time to expand a single node for each method is tabulated. Table 11 also presents a mean rank order of performance in

TABLE 11
MEAN PERFORMANCE

Strategy	Mean Node Expansion Time (sec)	Standard Deviation	Mean Order (1 - 6)	Standard Deviation
Unidirectional	0.0008	0.000081	5.2	0.45
Bidirectional	0.0008	0.000060	4.0	1.22
Heuristic-Selection	0.0008	0.000065	2.4	1.14
Ellipse	0.0008	0.000091	1.4	0.54
A*	0.0040	0.000327	3.2	0.71
$A_\epsilon{}^*$	0.0163	0.016738	4.8	2.17

which the ellipse method (bidirectional without heuristics) rates as the best while the unidirectional strategy is the worst performer. The table also presents standard deviation information. Based on this data, the $A_\epsilon{}^*$ method is the least-consistent method, both in time to expand a single node and in mean rank order. We note that the sample size used here is very small. However, the problems have been chosen to represent a wide class of typical problems and thus should be generally indicative of strategy performance.

E. SUMMARY

Table 10 shows that the bidirectional, heuristic-selection and ellipse methods all have low overhead costs, comparable to that of the unidirectional strategy. The ellipse method is the best overall performer. The heuristic-selection method is occasionally good and when it does not speed the search, there is no performance decrease. This statement cannot be made regarding the A* and $A_\epsilon{}^*$ strategies. Their high overhead is detrimental in some cases. These results confirm that low-overhead exhaustive strategies are appropriate for wavefront-propagation techniques.

The results of Table 10 and 11 indicate that combining the ellipse and heuristic-selection methods has advantages. That is, we use an ellipse to impose a global limit on the problem and use heuristic-selection to constrain wavefront growth within the ellipse. This method is compared against the standard ellipse and heuristic-selection techniques in the problems depicted in Figures 44 through 52. The performance of each method is tabulated in Table 12.

TABLE 12
PERFORMANCE COMPARISON

Strategy	Problem Number	Figure Number	CPU Time (seconds)	Nodes Expanded	Order of Performance
Heuristic-Selection	F	44	8.25	9832	3
	G	47	18.18	23397	3
	H	50	28.69	32884	2
Ellipse	F	45	7.97	9250	2
	G	48	15.63	19776	2
	H	51	30.15	35476	2
Ellipse & Heuristic-Selection	F	46	7.61	9064	1
	G	49	14.41	19040	1
	H	52	20.33	25158	1

Note that the solutions presented in Figures 47 and 48 differ from the solution of Figure 49. This is a result of solution path cost error. In Figures 47 and 48, the wavefronts touch interior to approximating chords, the maximum-error situation. In Figure 49, the wavefronts touch at chord endpoints, the minimum-error case. Also note that the two separate paths are close in path cost. This can be seen by the proximity of the wavefronts in both places where solution wavefronts intersect (i.e., the lower portion of Figure 49 and in the upper portions of Figures 47 and 48).

We note that none of the methods discussed in this chapter lower the $O(n^2)$ worst-case complexity of wavefront propagation. (Where n is the number of lattice nodes.) However, the methods listed in Table 12 improve the average-case performance of unidirectional wavefront propagation without degrading it in the worst case. Of all the methods, heuristic-selection within an ellipse seems to have the lowest time requirements. Also, this method requires only a small increase in storage space. However, it does not cure the inherent problems of wavefront propagation. The solution paths offered by wavefront propagation are inaccurate in terms of path cost (a topic more fully developed in Chapters VI and VII). The inaccuracies stem from two resolution-dependent aspects inherent in the problem representation. The first depends on the number of nodes in the lattice. The second is determined by the connectivity, or branching factor at each node in the lattice. The development and usage of a more appropriate problem representation for the weighted-region problem is the subject of the following chapters.

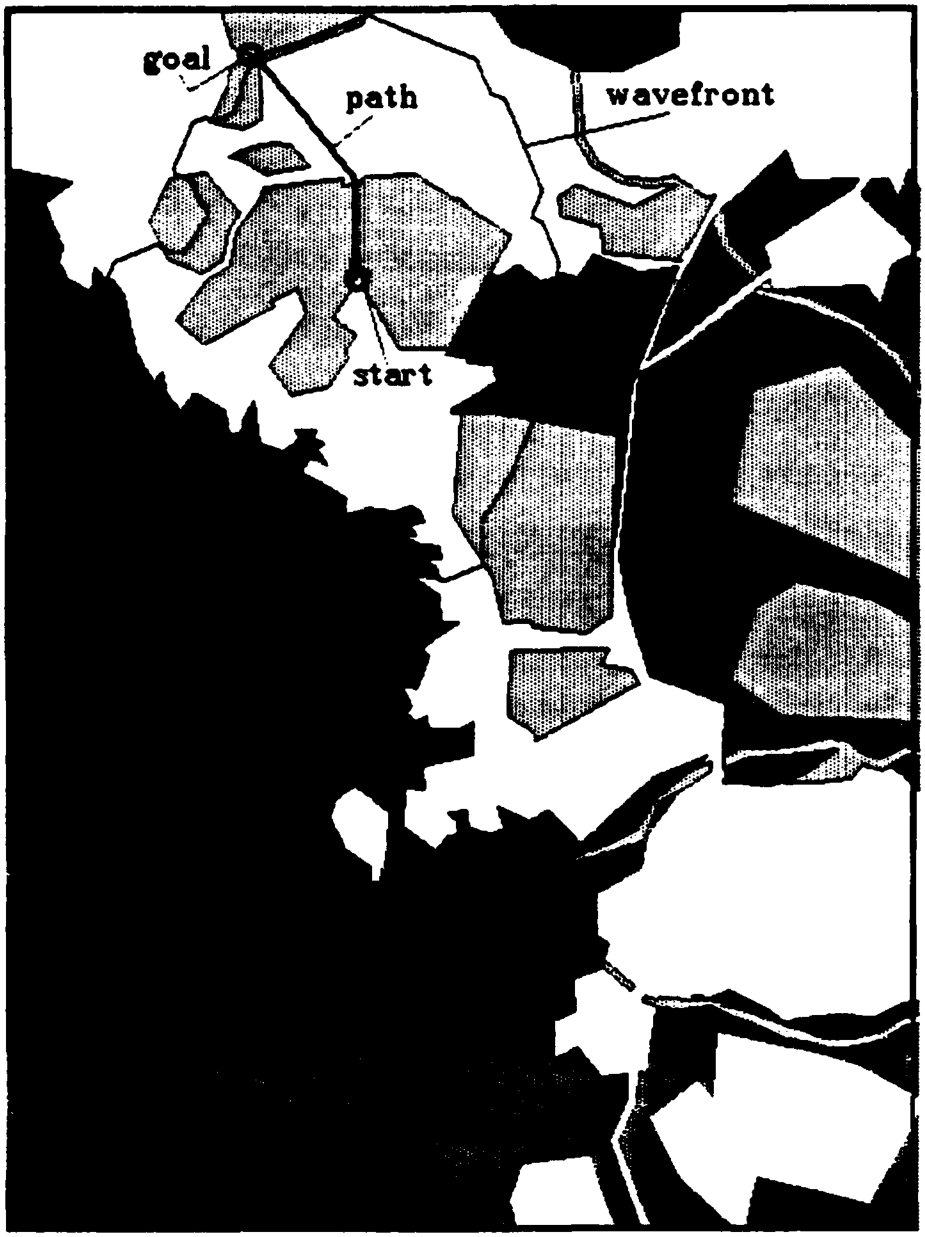


Figure 14. Problem A. Unidirectional Strategy

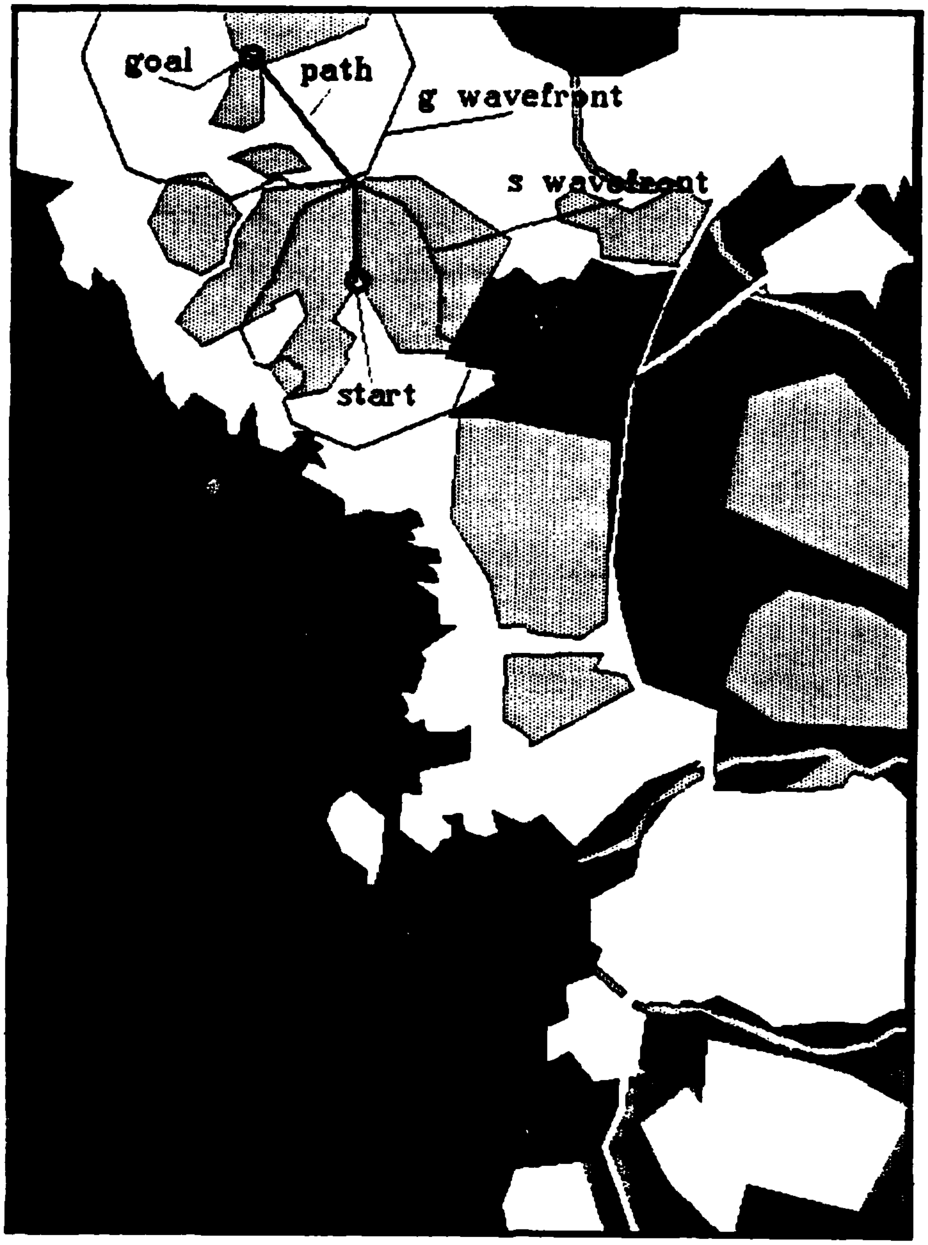


Figure 15. Problem A, Bidirectional Strategy

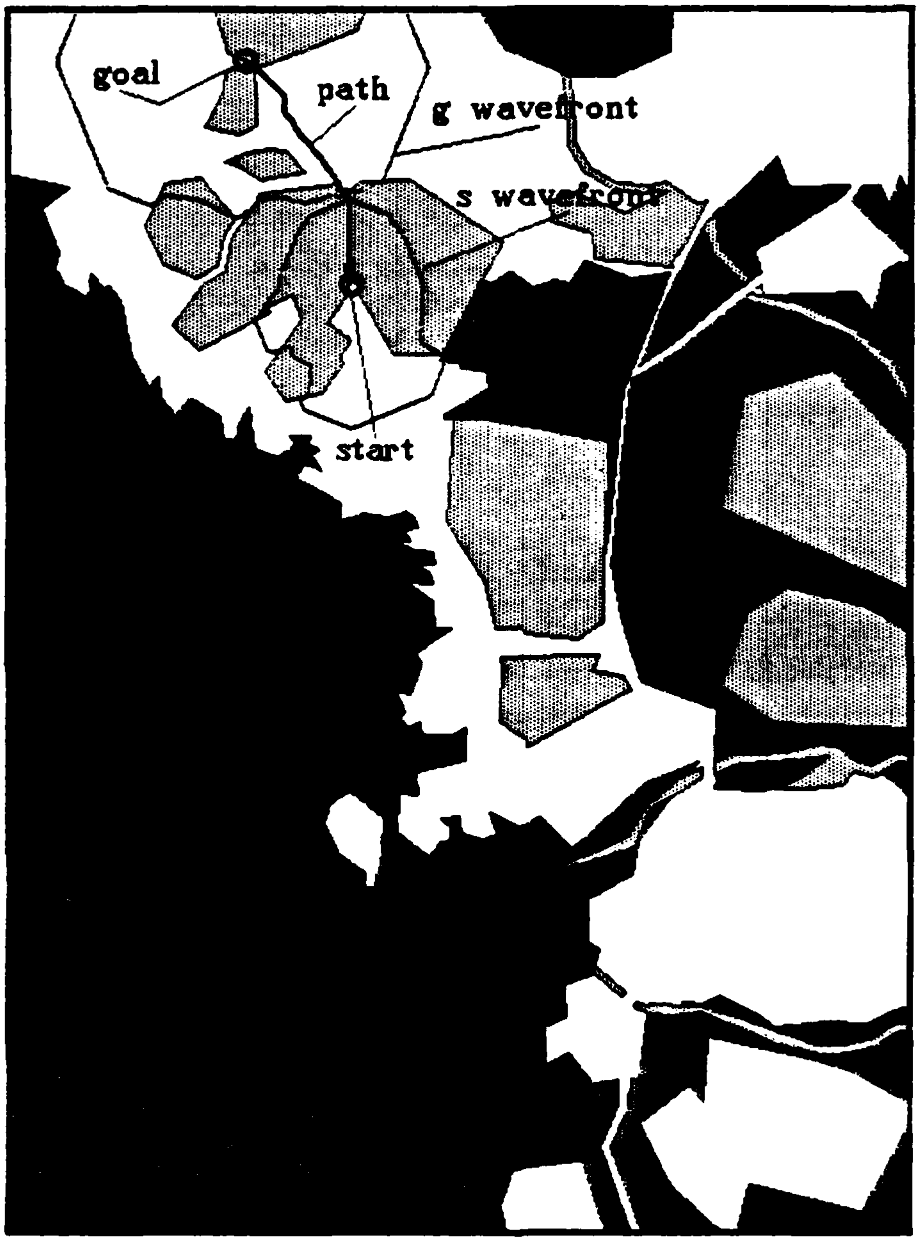


Figure 16. Problem A. Heuristic-Selection Strategy

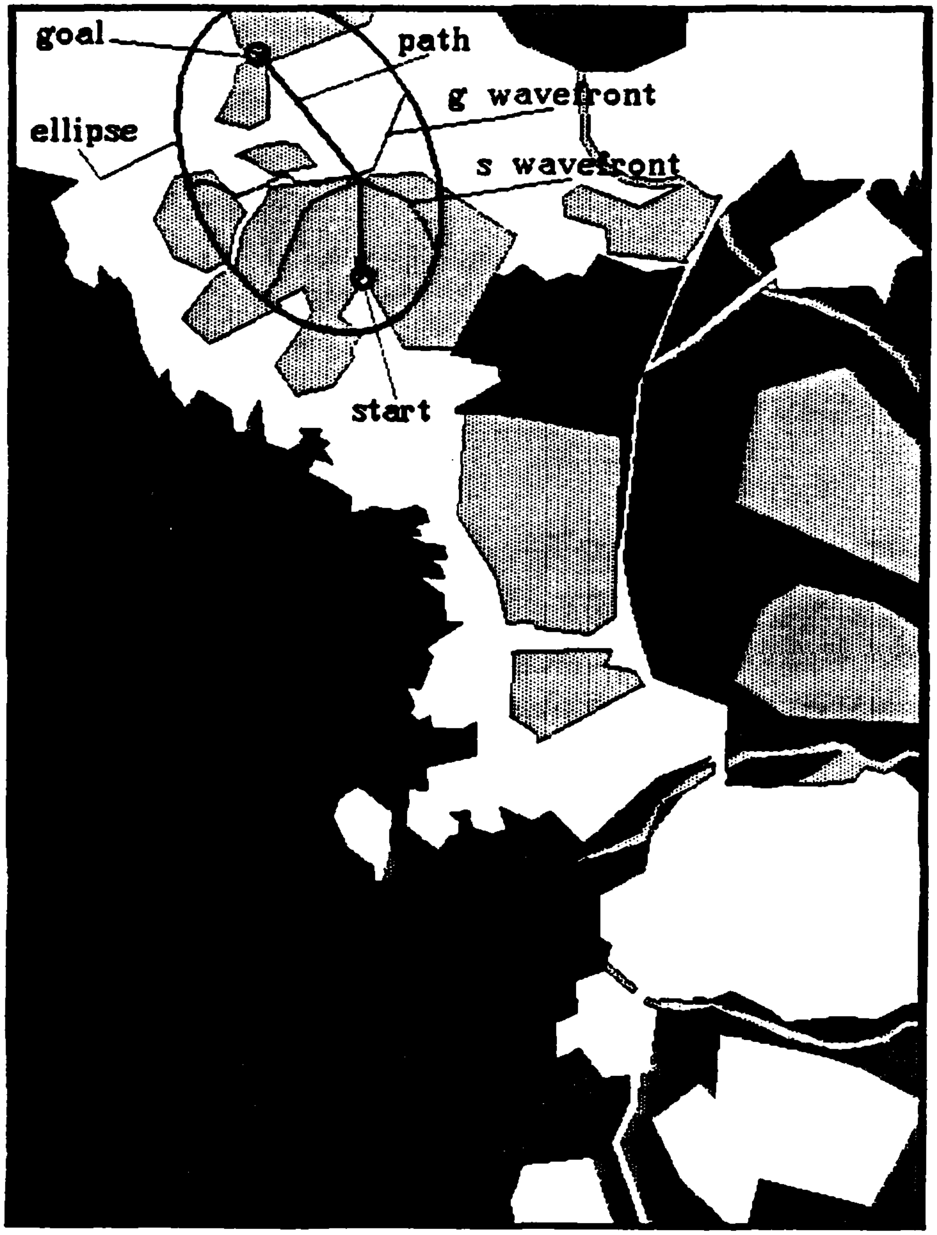


Figure 17. Problem A. Ellipse Strategy

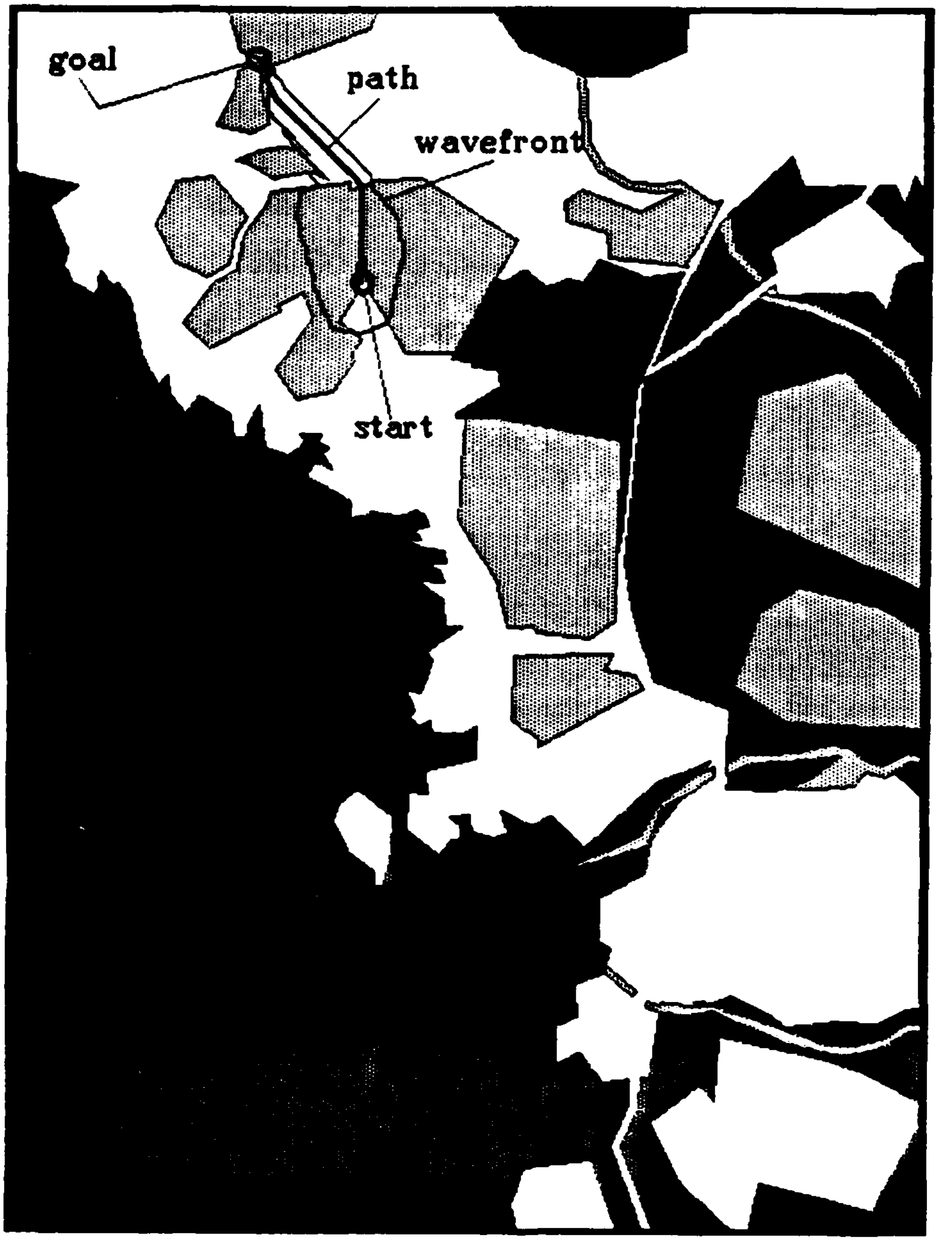


Figure 18. Problem A. A* Strategy

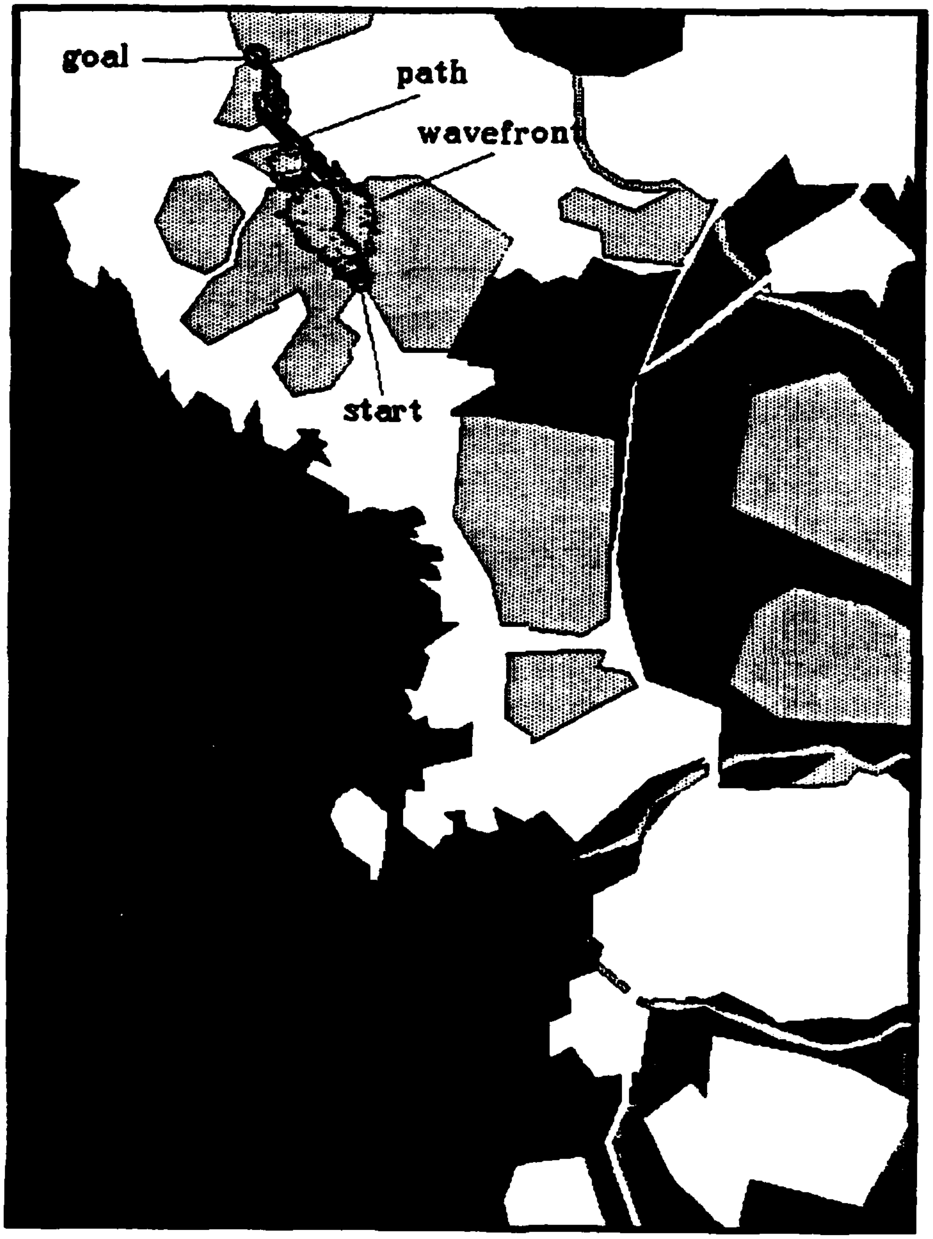


Figure 19. Problem A. $A_\epsilon{}^*$ Strategy

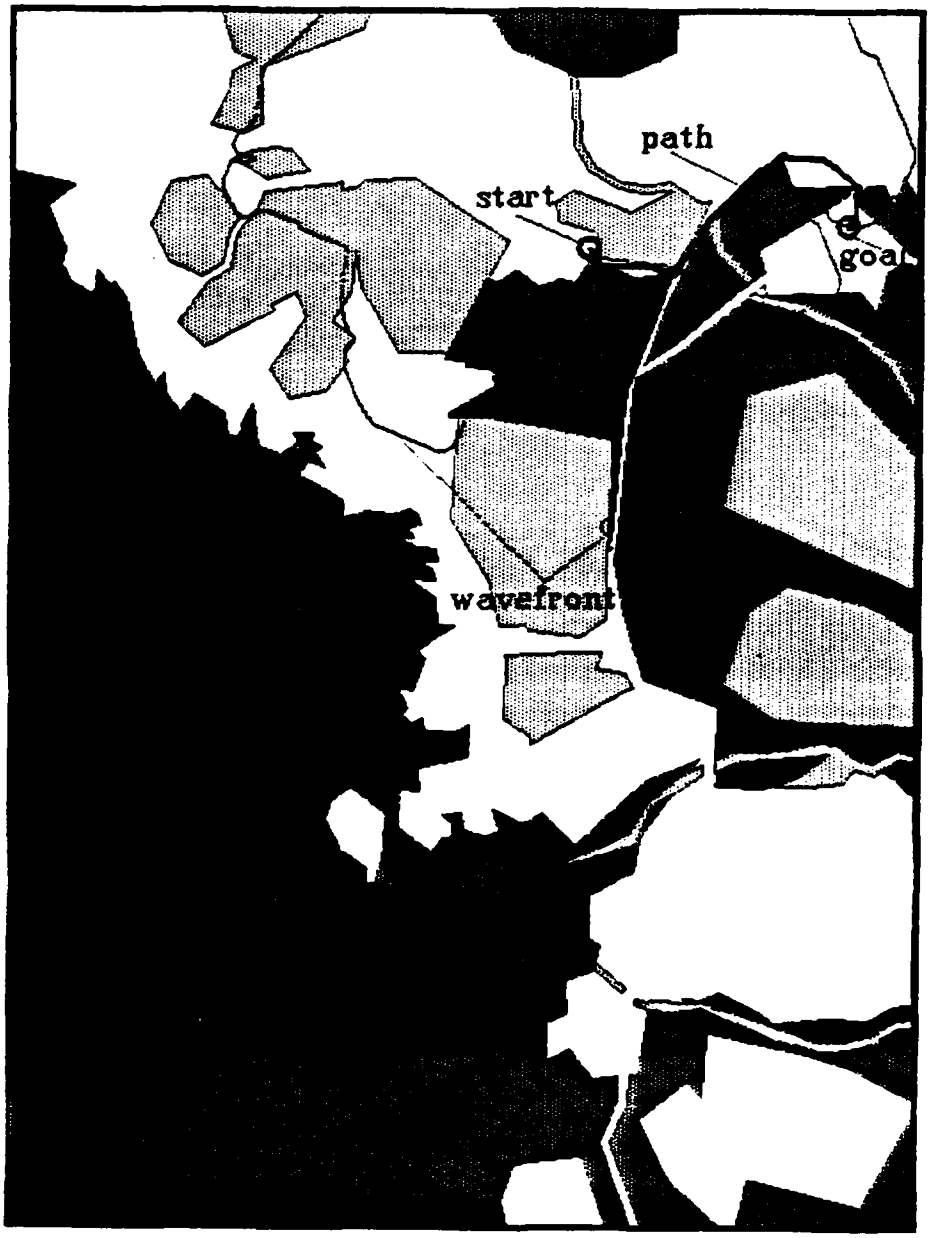


Figure 20. Problem B, Unidirectional Strategy

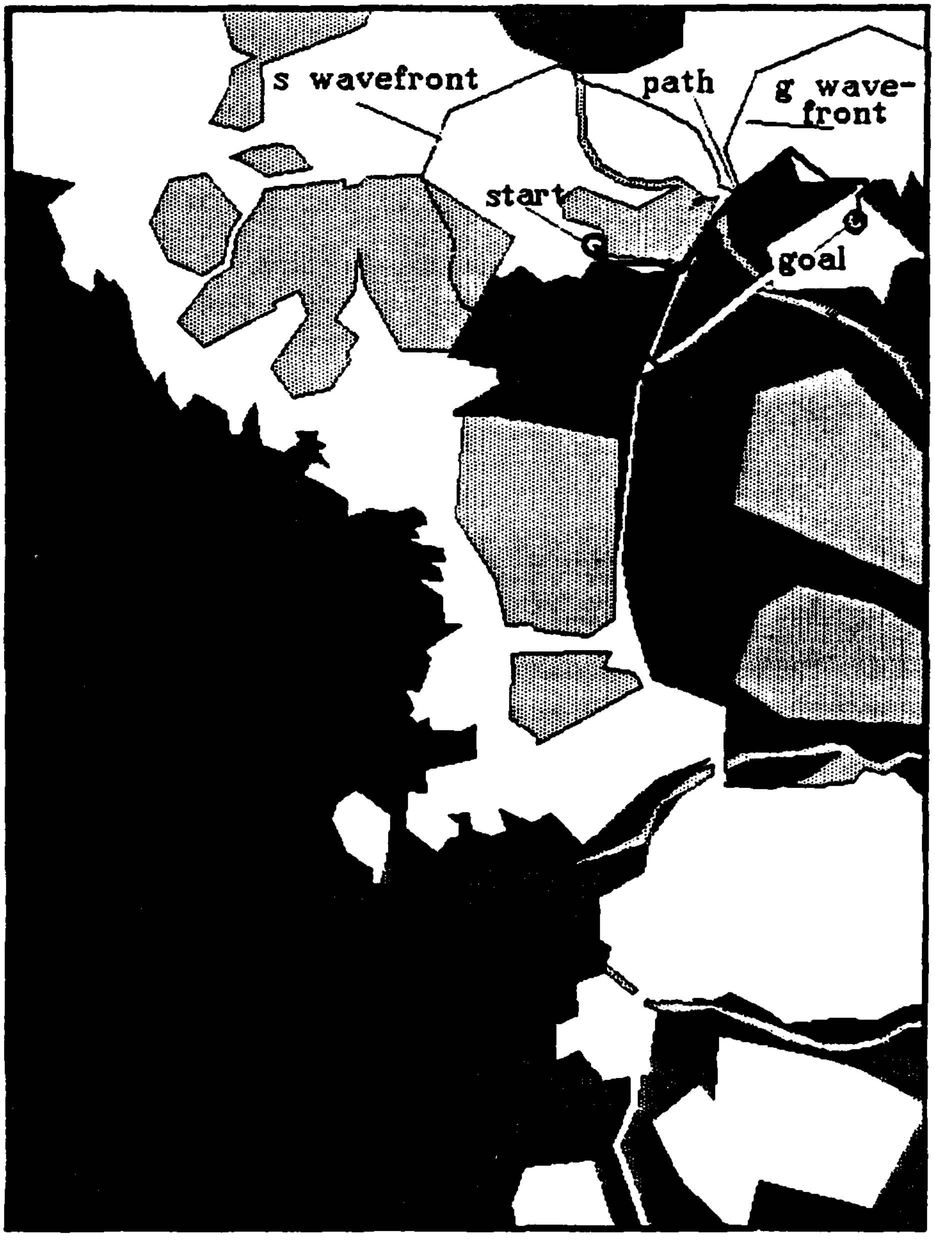


Figure 21. Problem B. Bidirectional Strategy

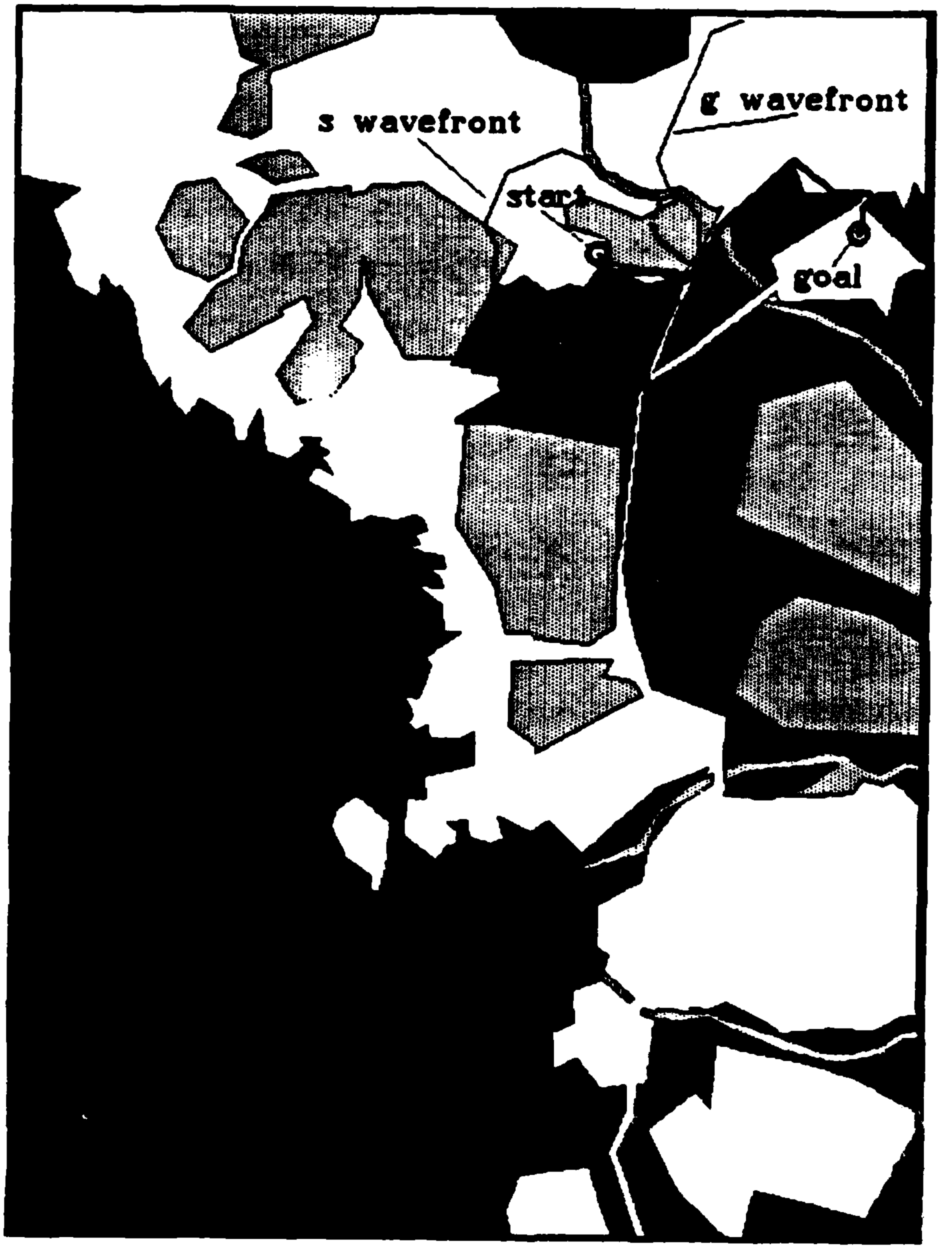


Figure 22. Problem B, Heuristic-Selection Strategy

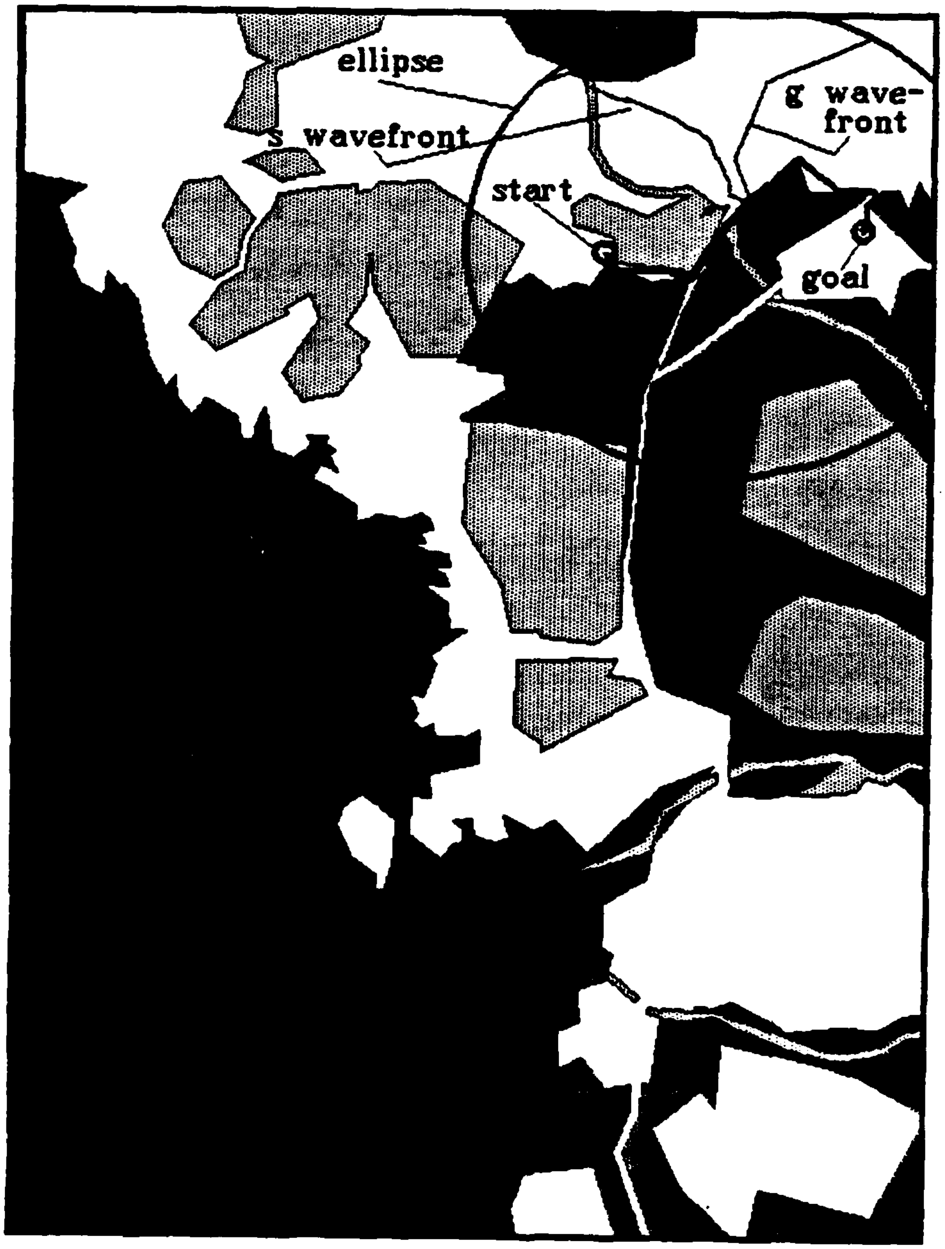


Figure 23. Problem B, Ellipse Strategy

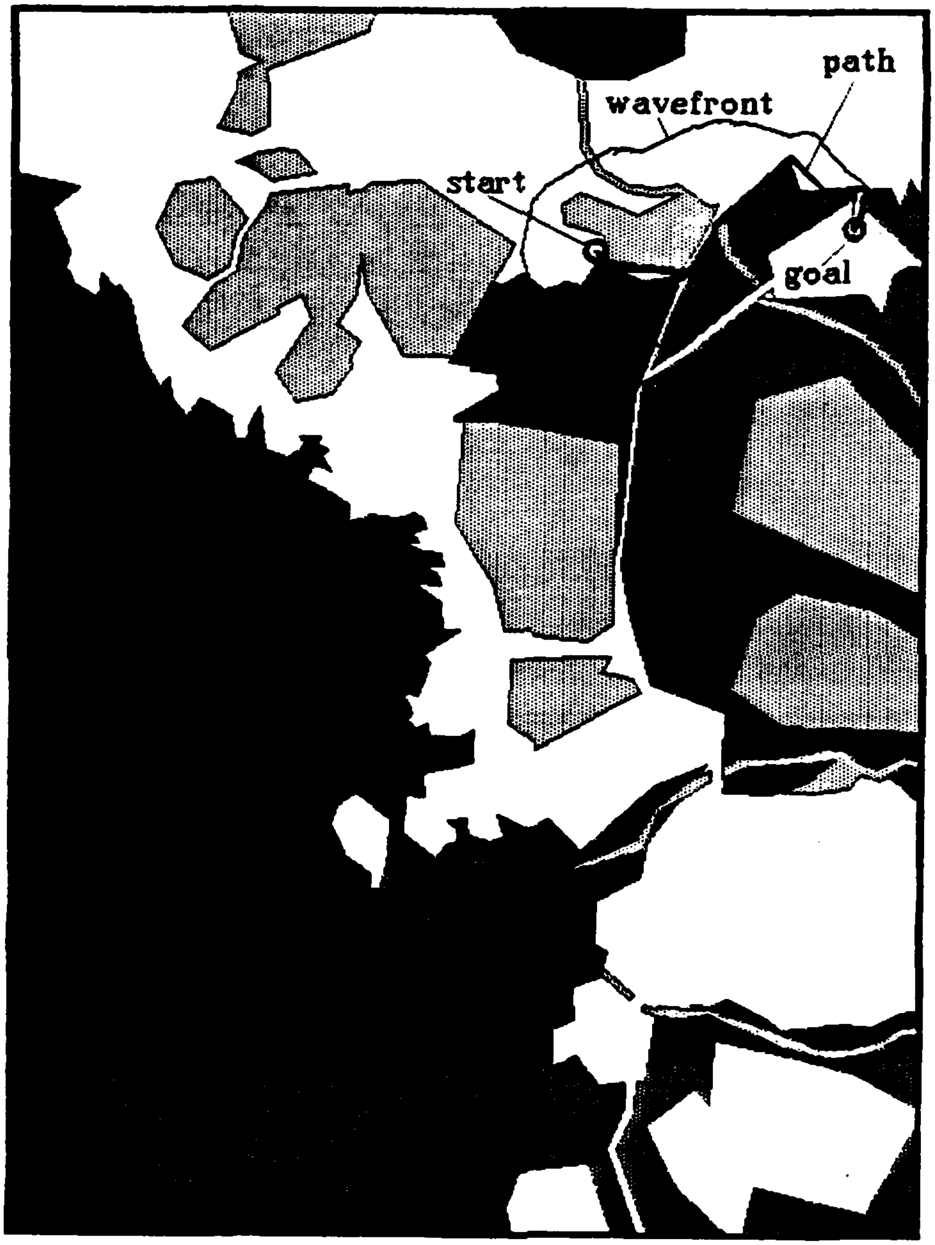


Figure 24. Problem B. A* Strategy

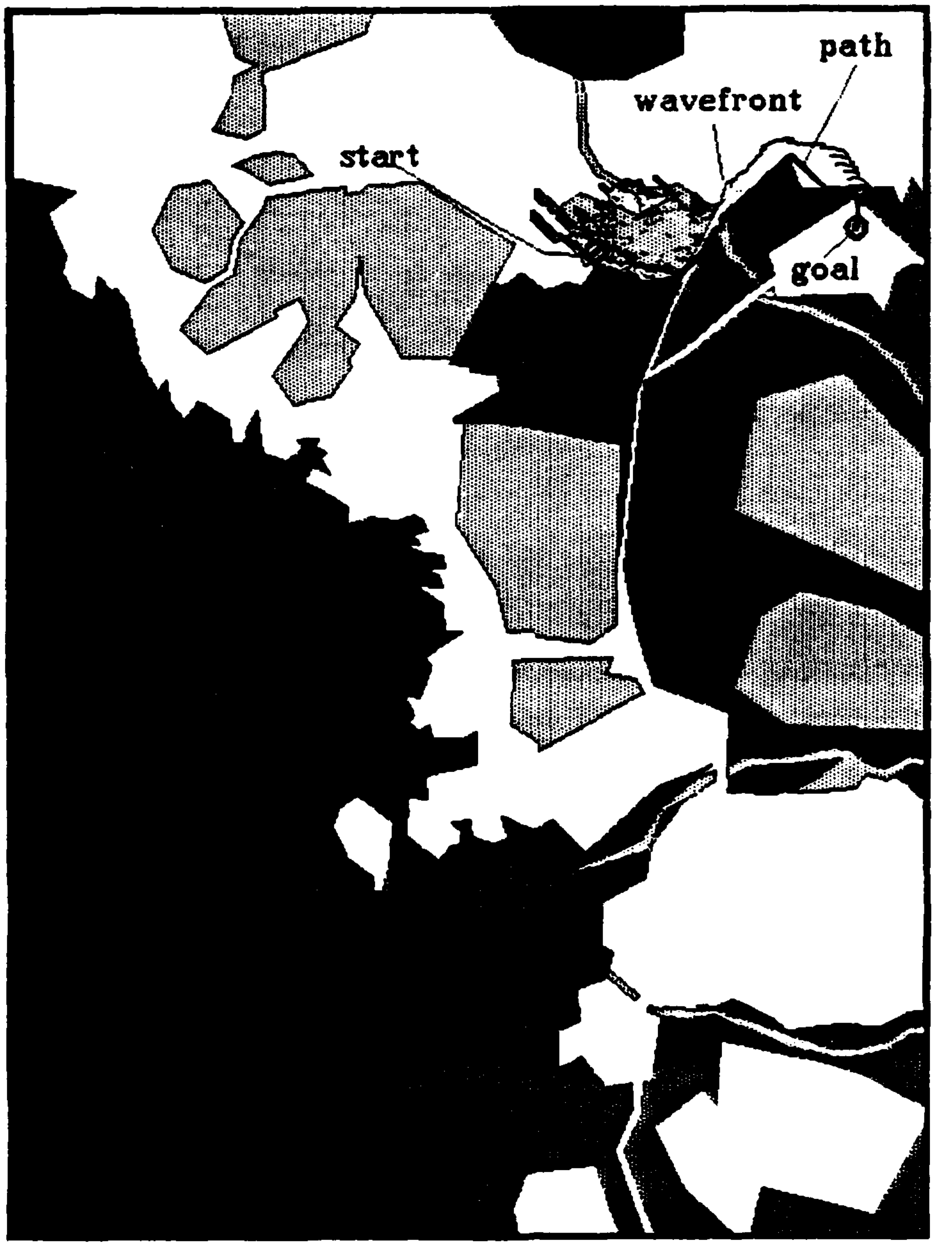


Figure 25. Problem B, $A_\epsilon{}^*$ Strategy

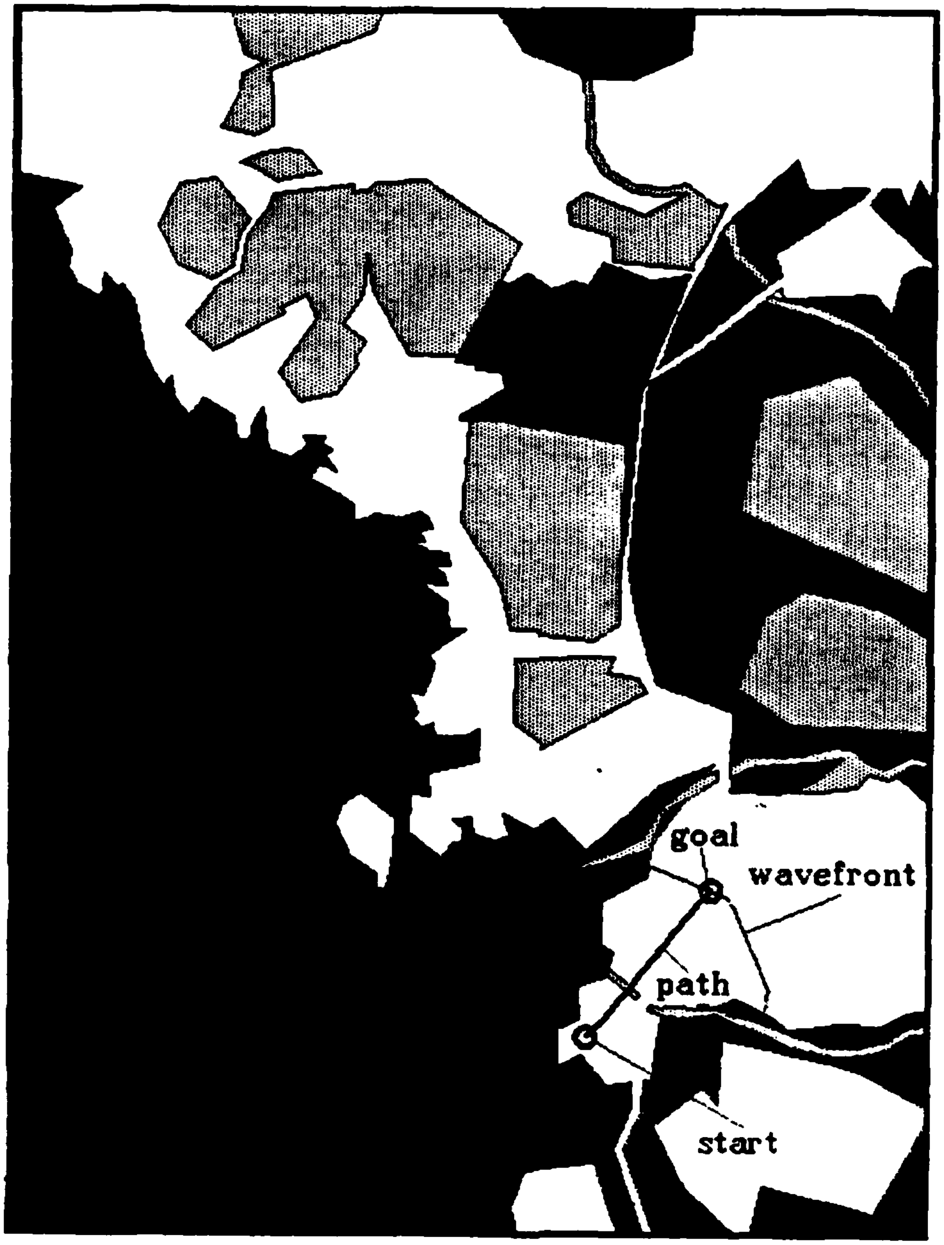


Figure 26. Problem C, Unidirectional Strategy

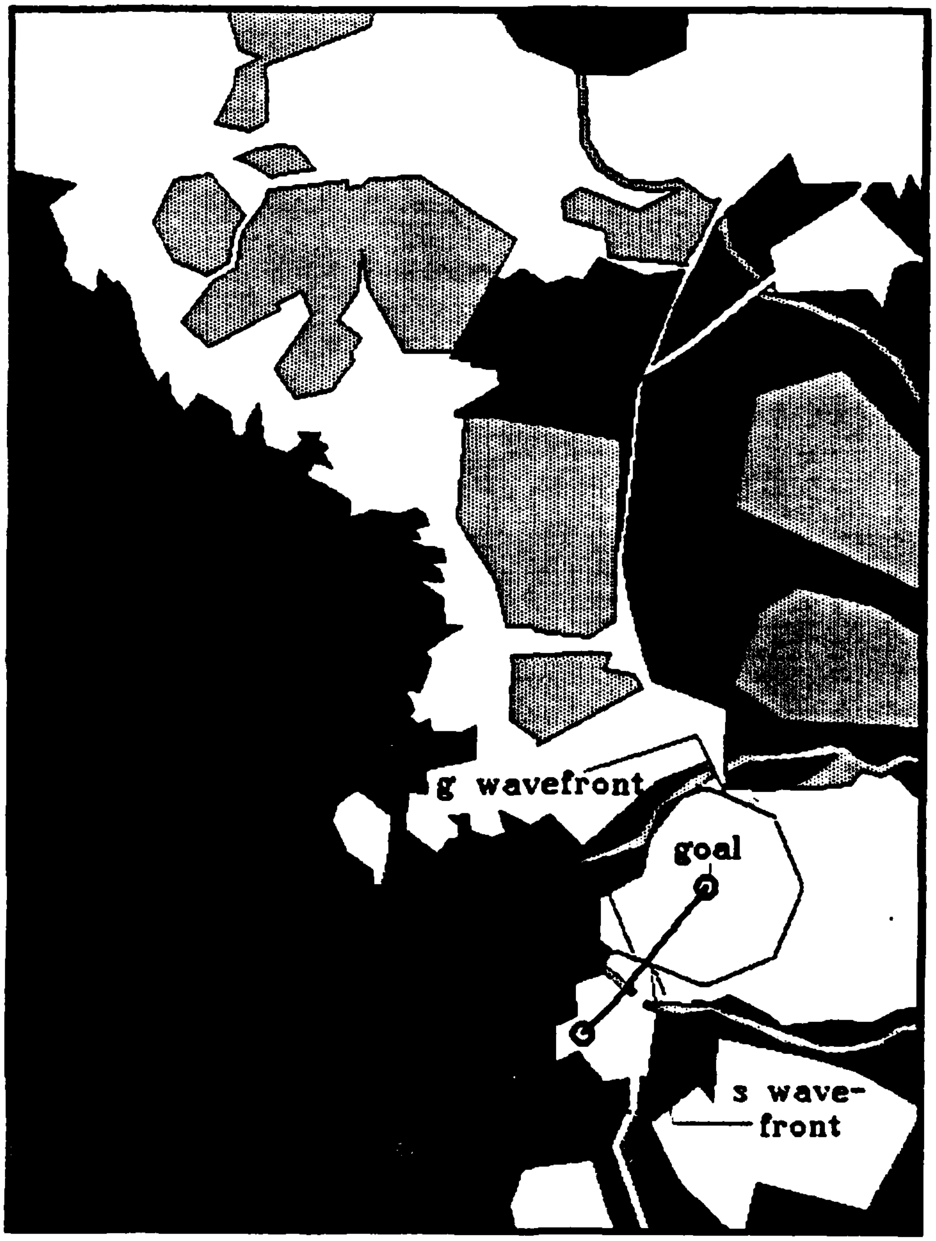


Figure 27. Problem C, Bidirectional Strategy

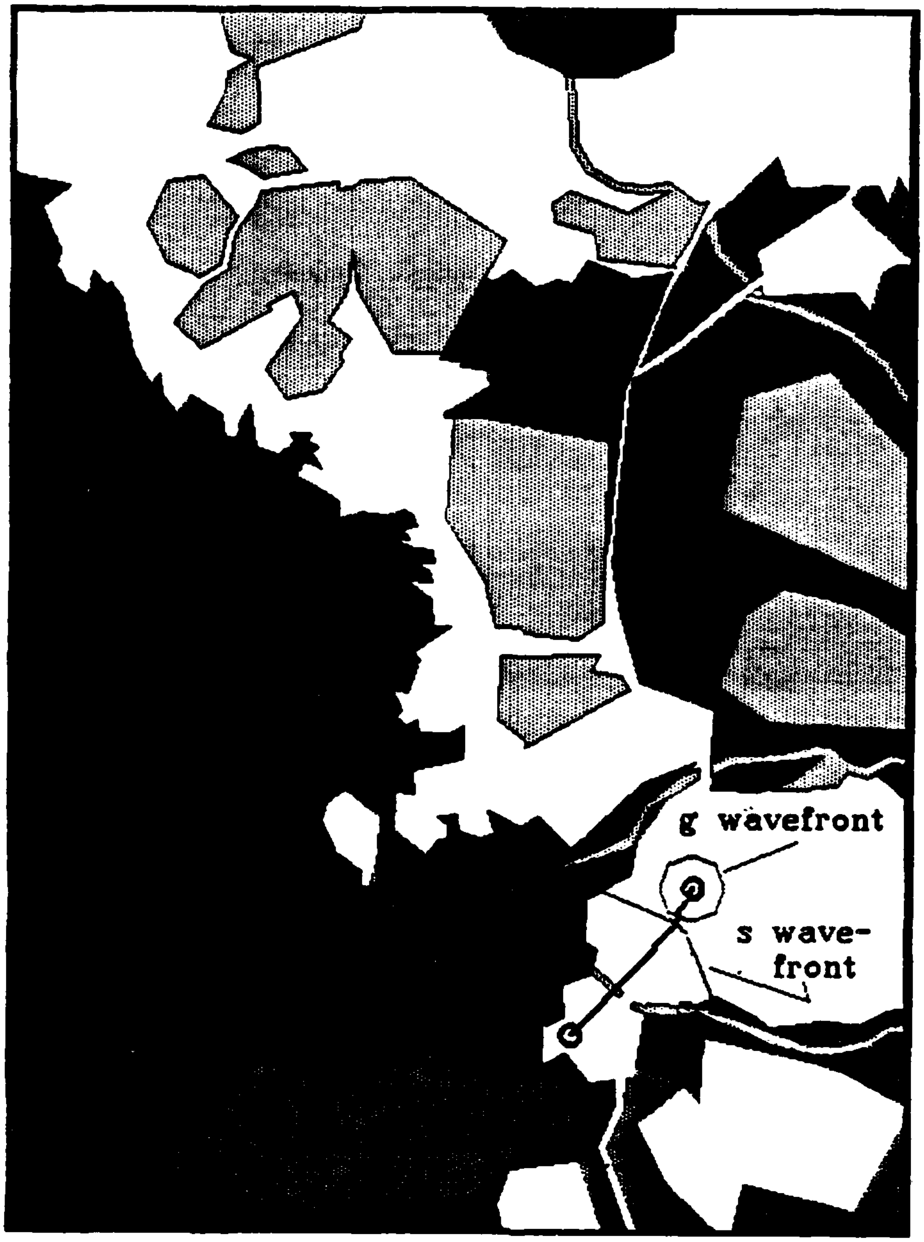


Figure 28. Problem C, Heuristic-Selection Strategy

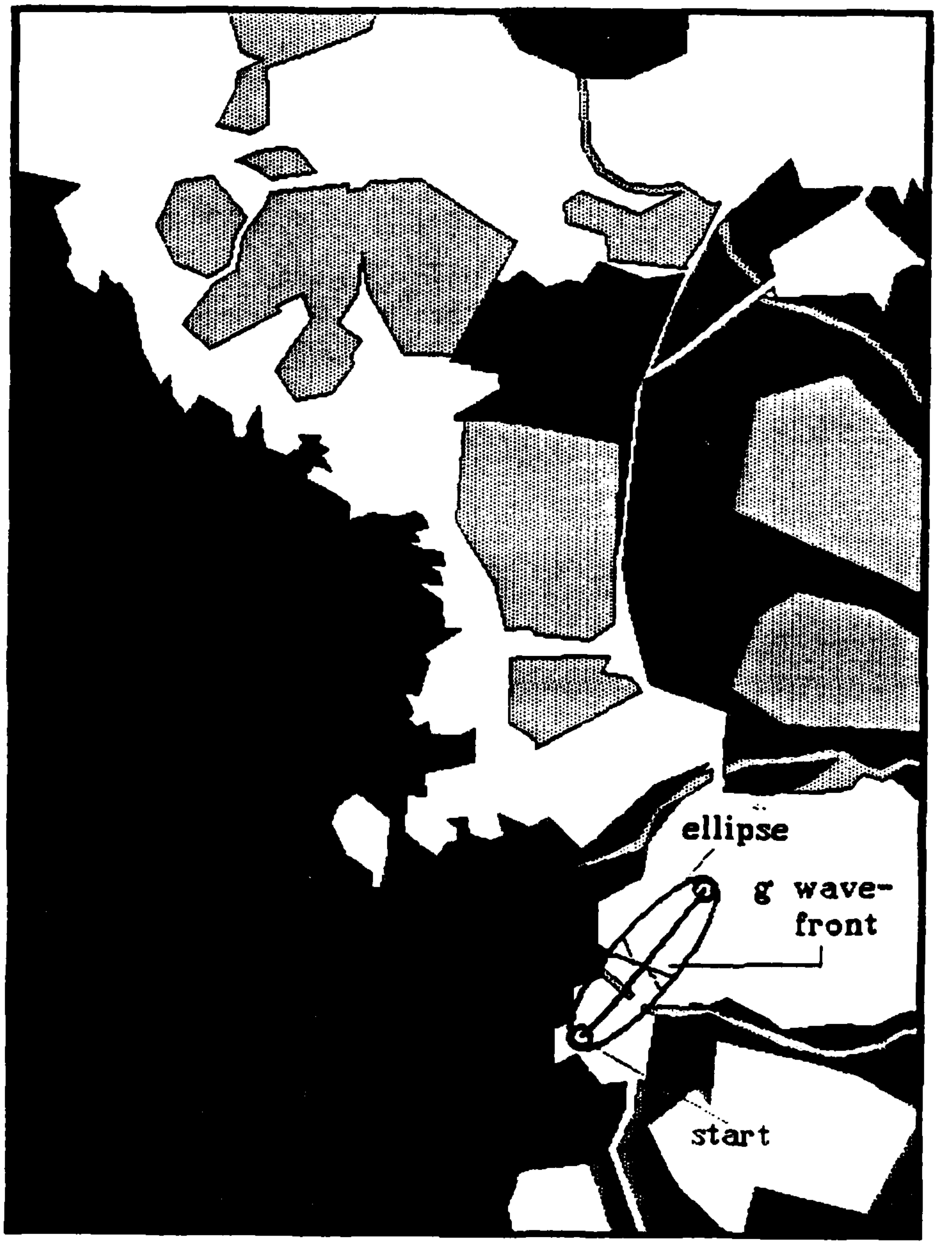


Figure 29. Problem C, Ellipse Strategy

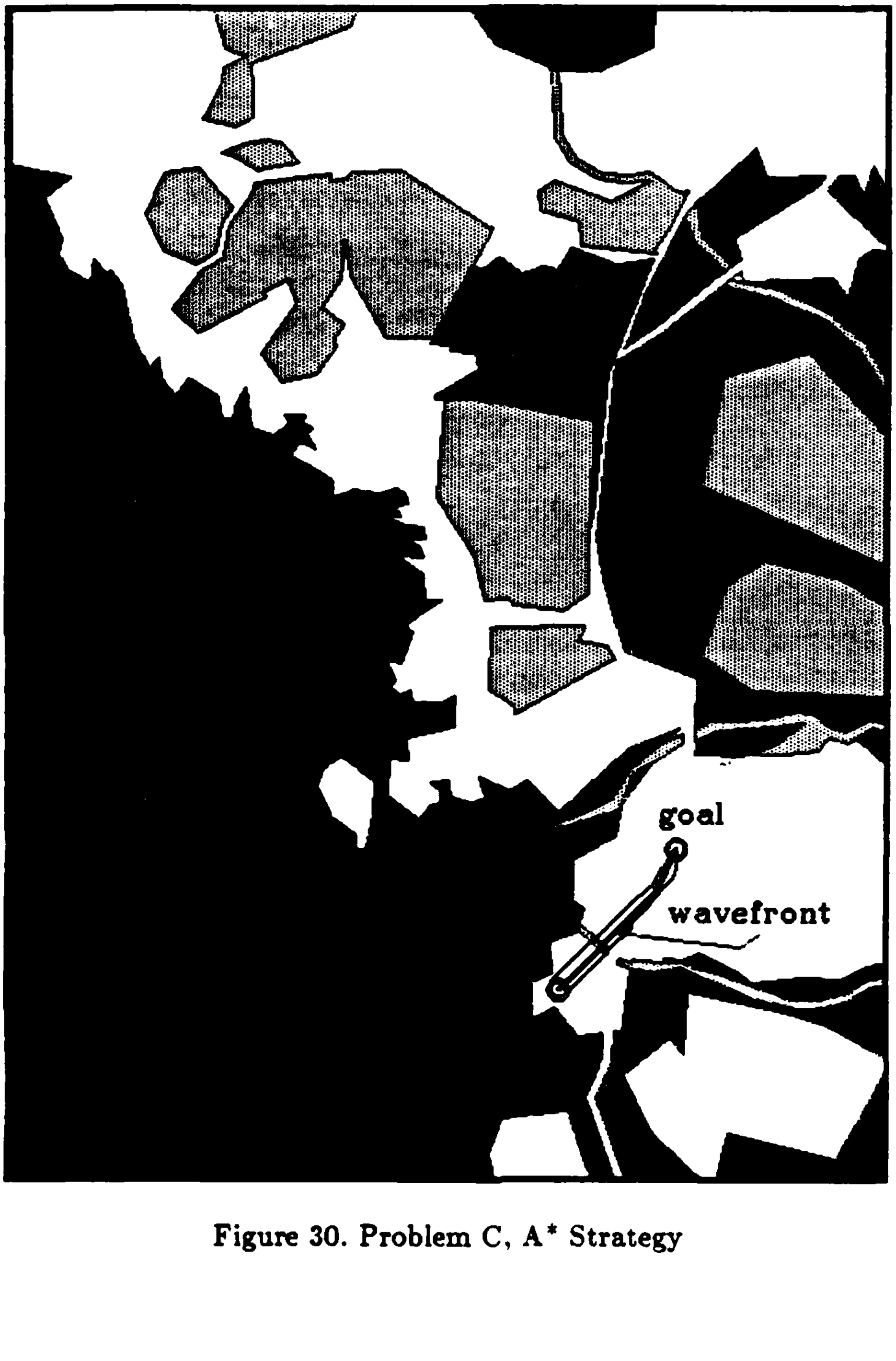


Figure 30. Problem C, A* Strategy

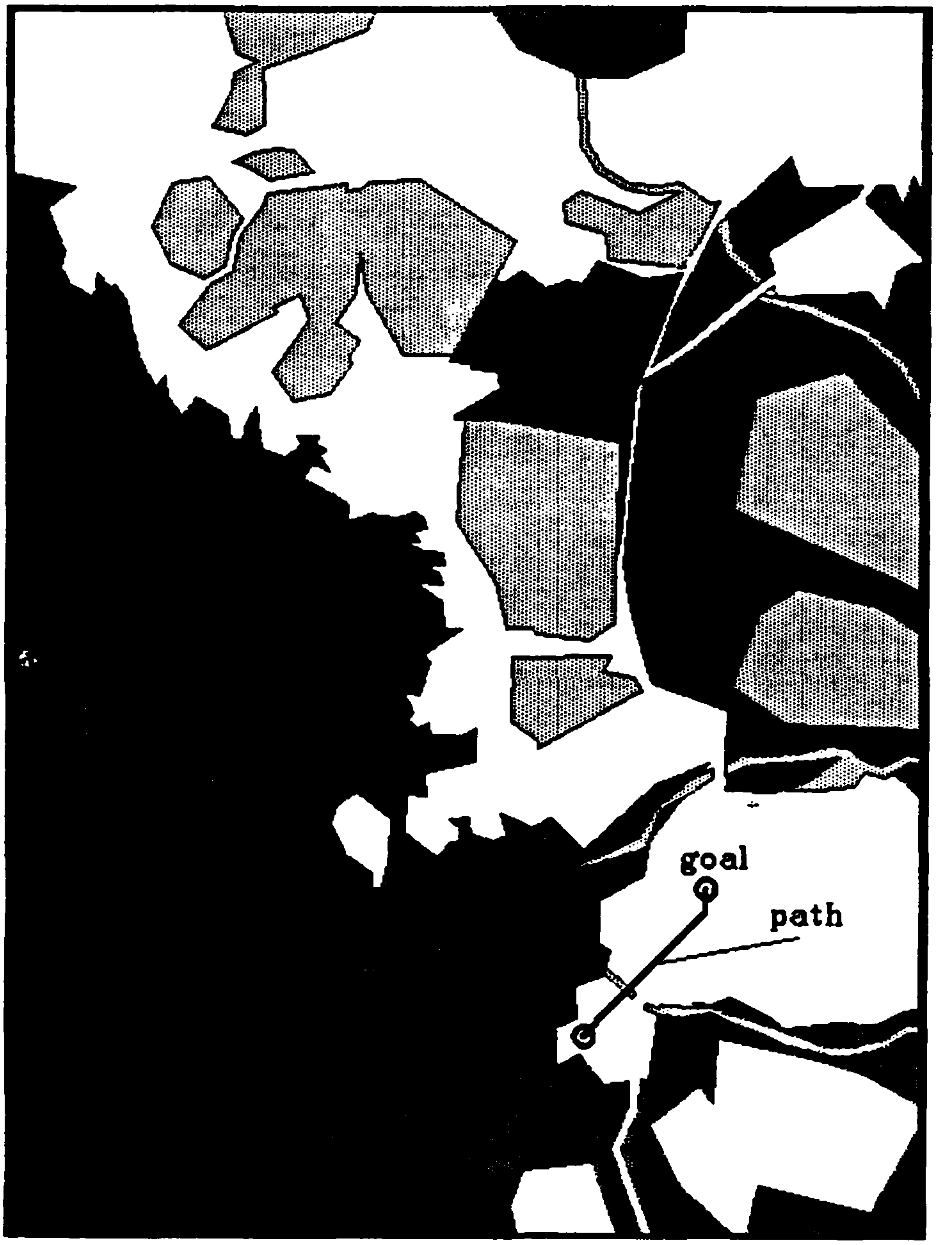


Figure 31. Problem C, A_ϵ* Strategy

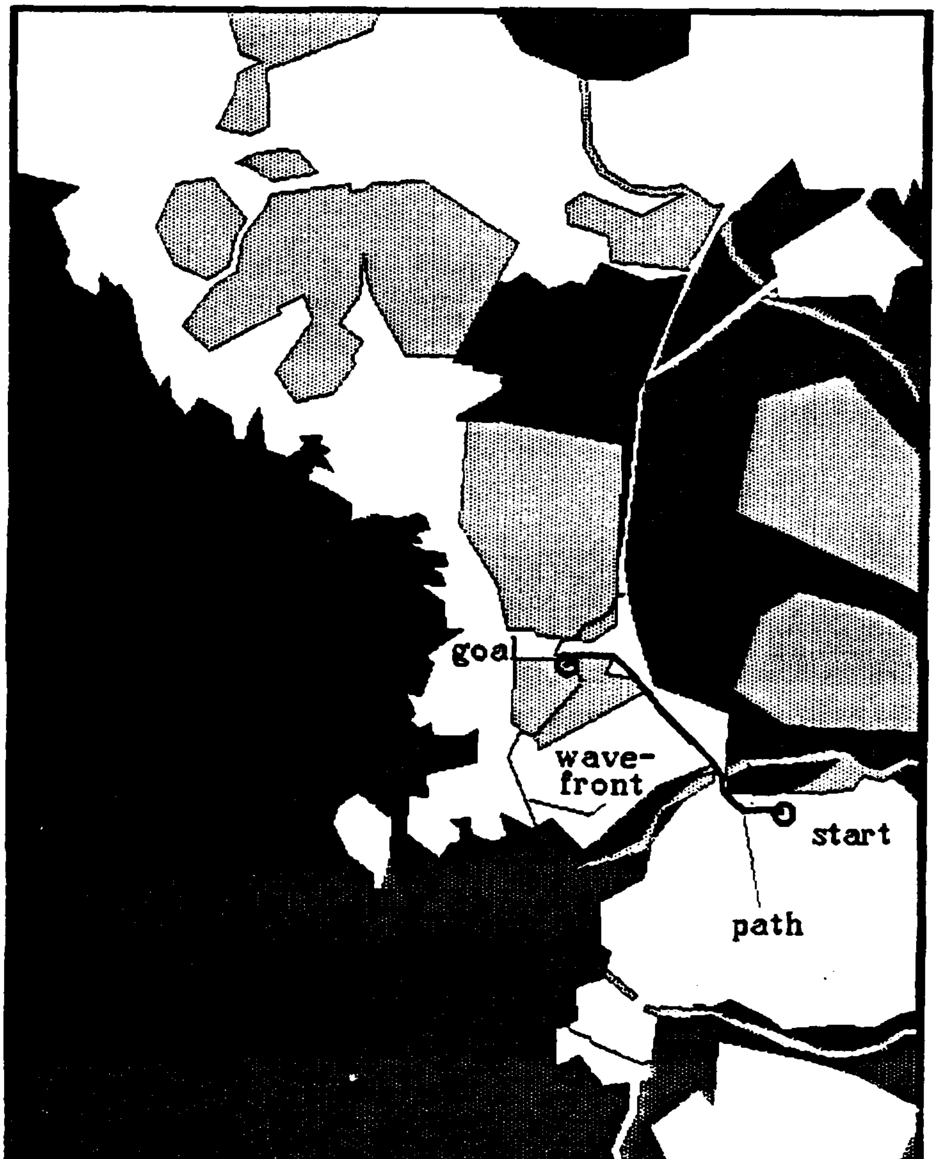


Figure 32. Problem D, Unidirectional Strategy

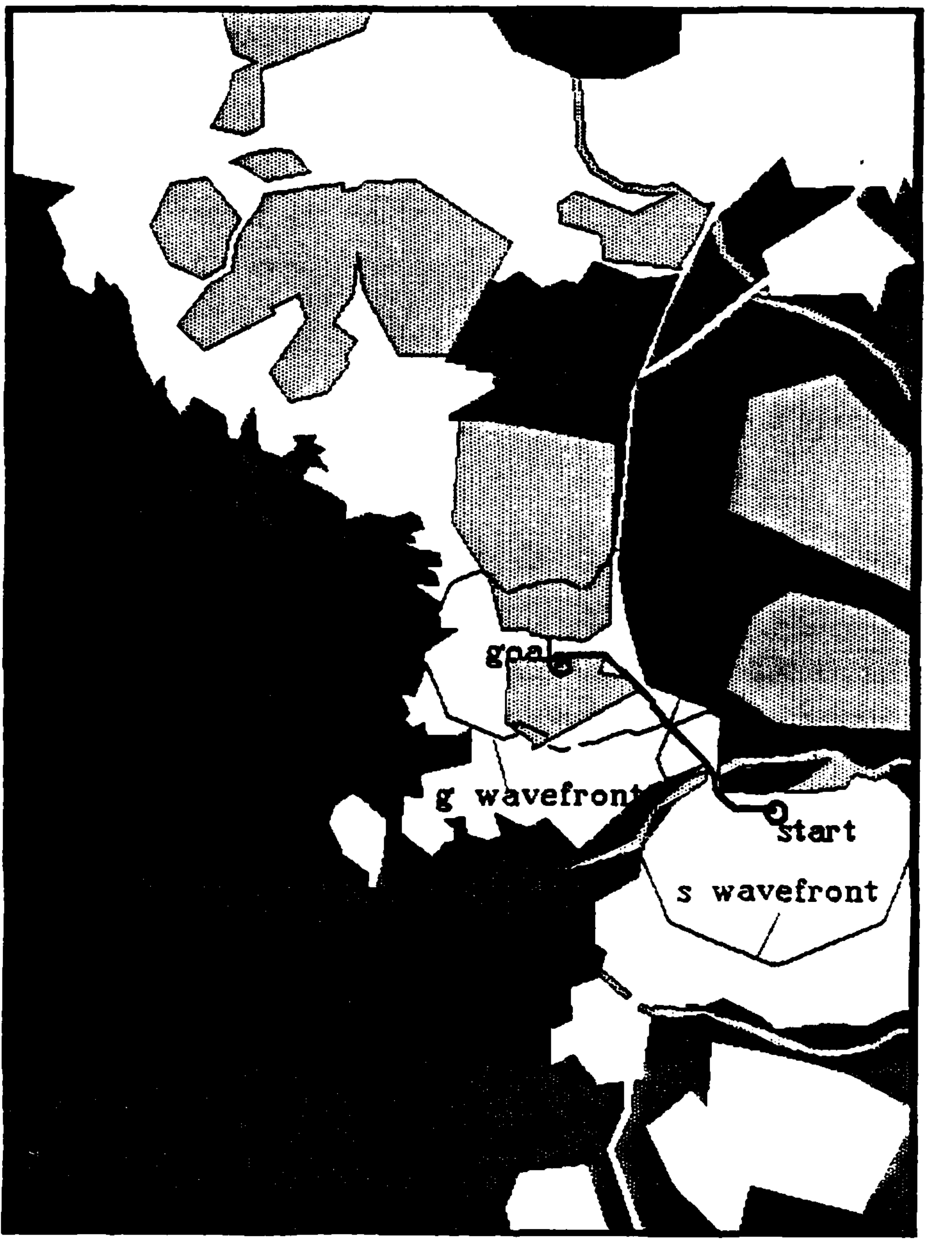


Figure 33. Problem D, Bidirectional Strategy

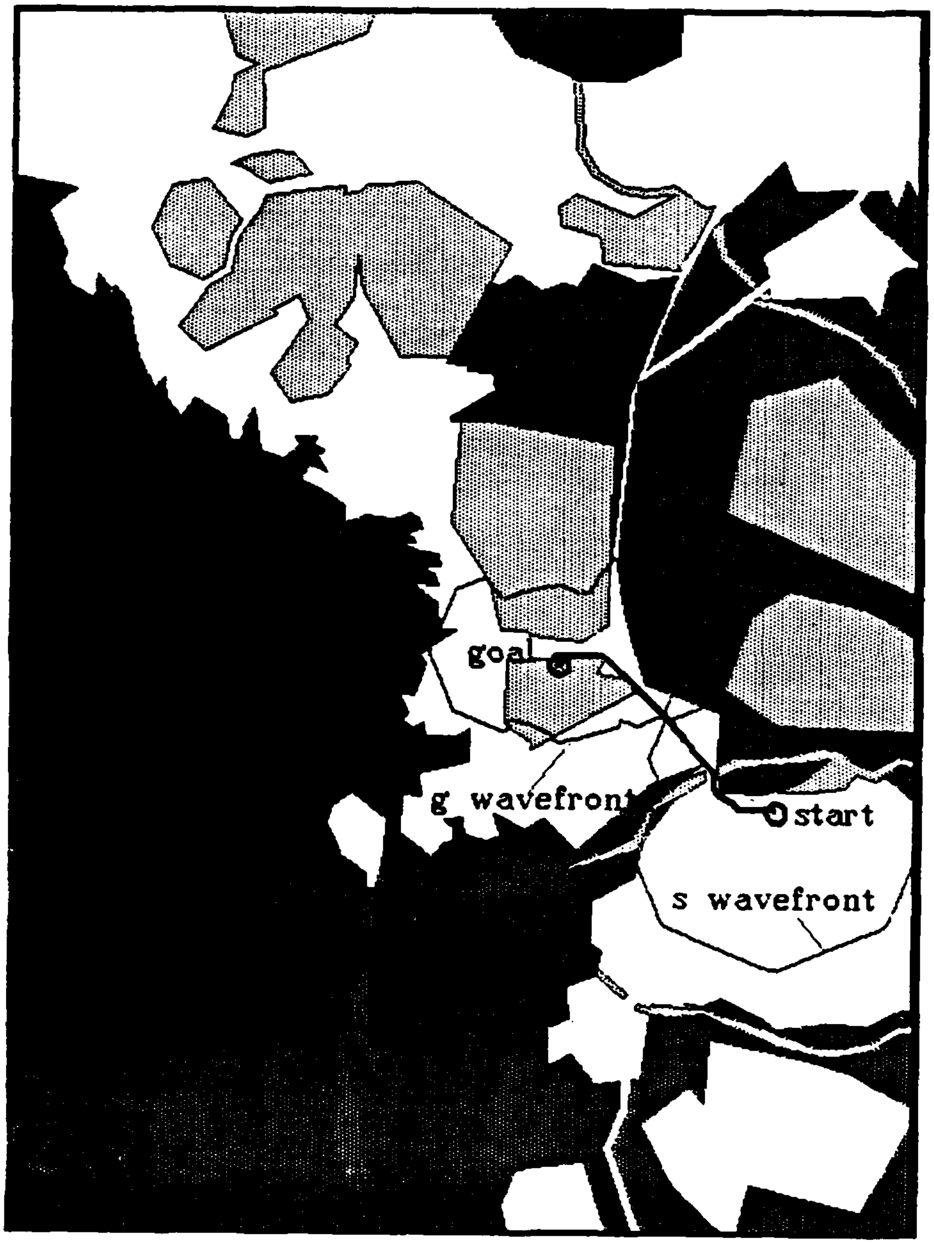


Figure 34. Problem D, Heuristic-Selection Strategy

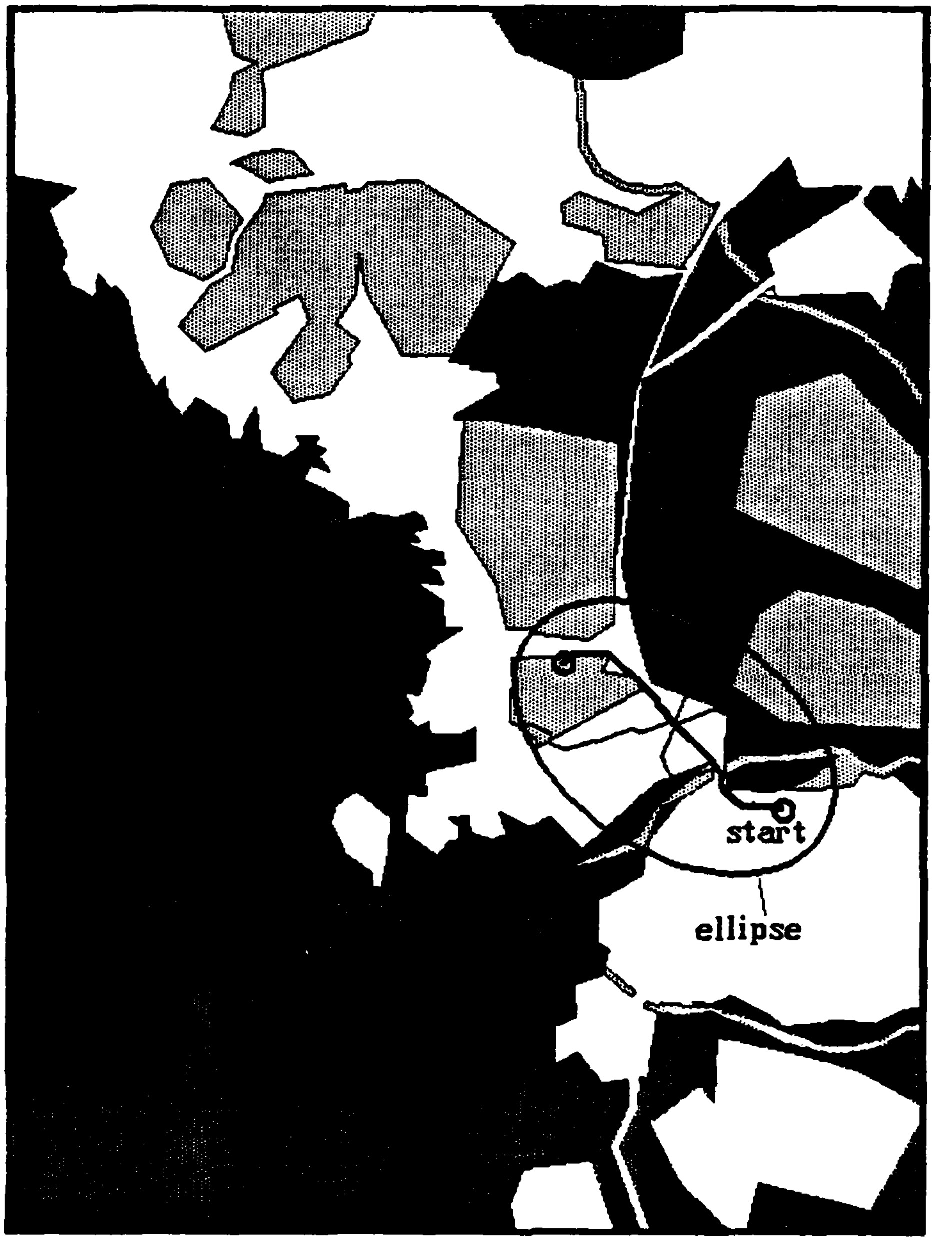


Figure 35. Problem D, Ellipse Strategy

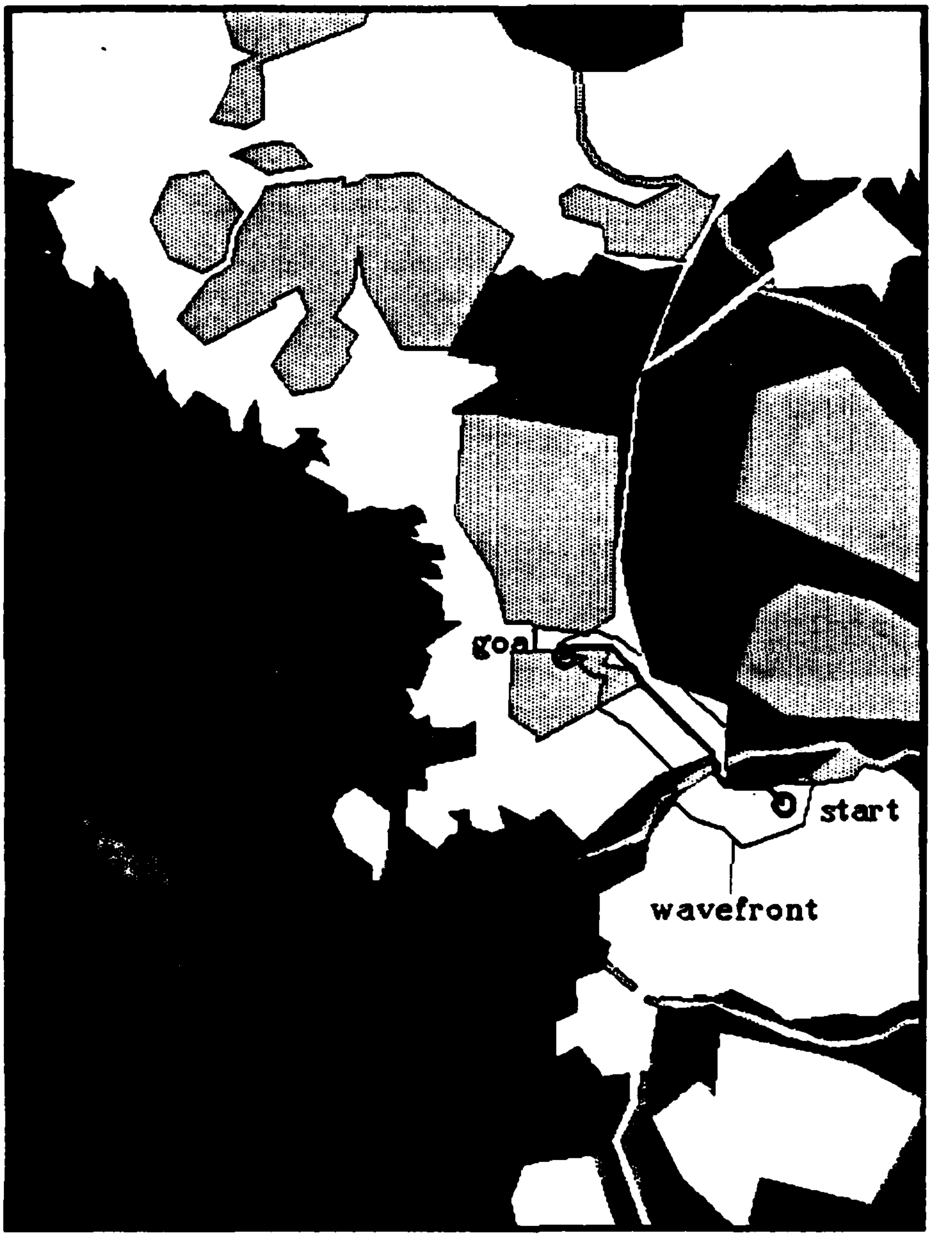


Figure 36. Problem D, A* Strategy

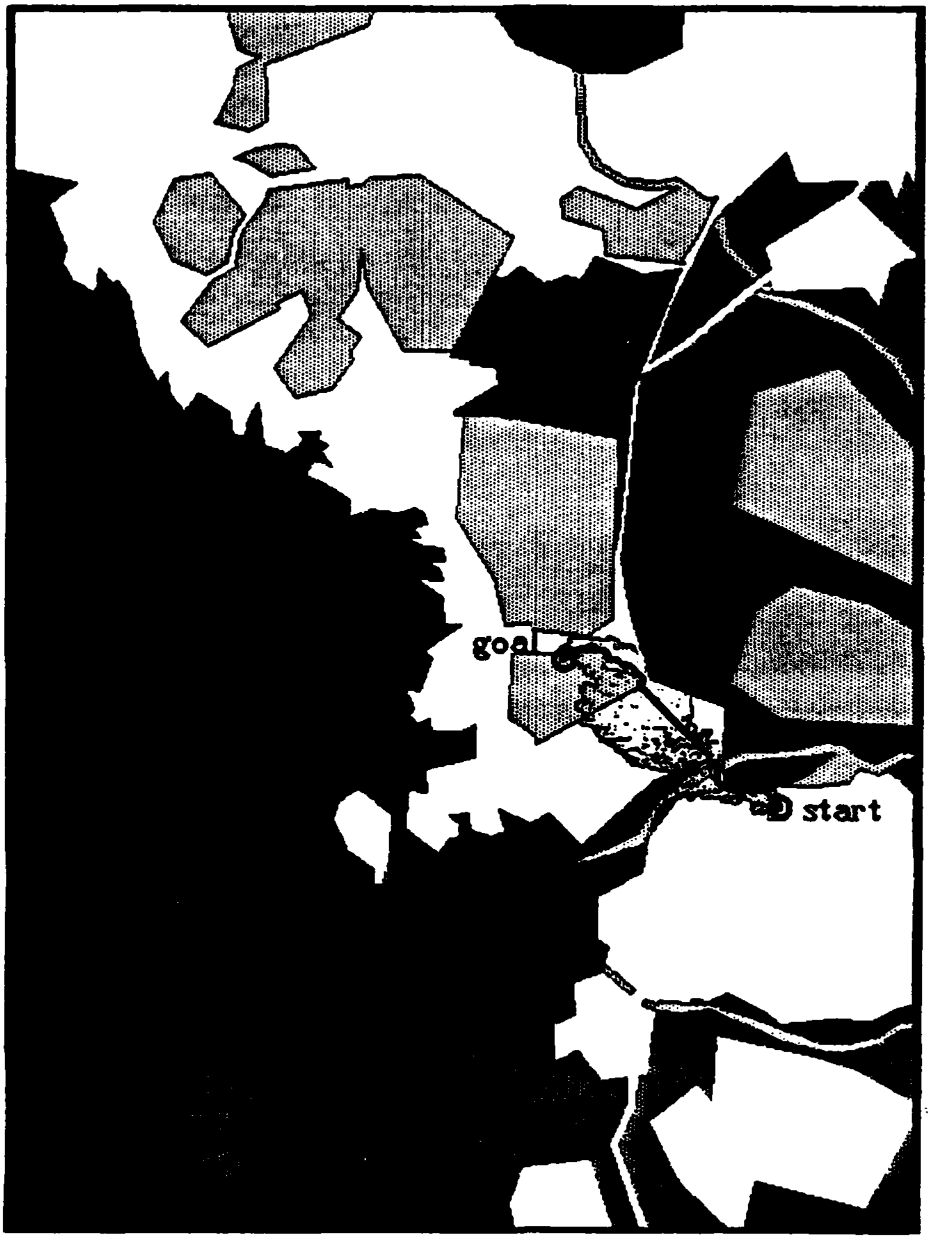


Figure 37. Problem D, A_{ϵ}* Strategy

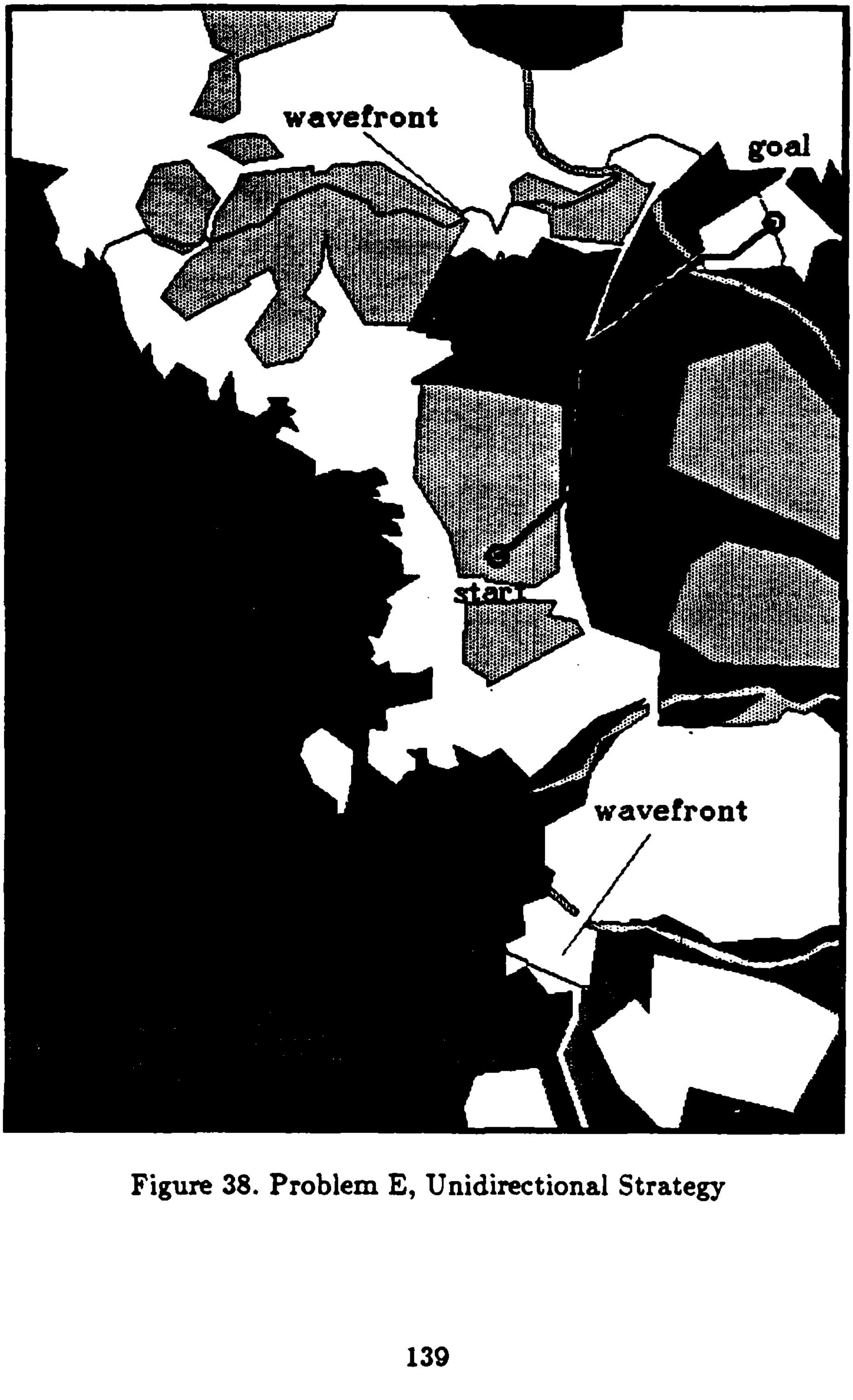


Figure 38. Problem E, Unidirectional Strategy

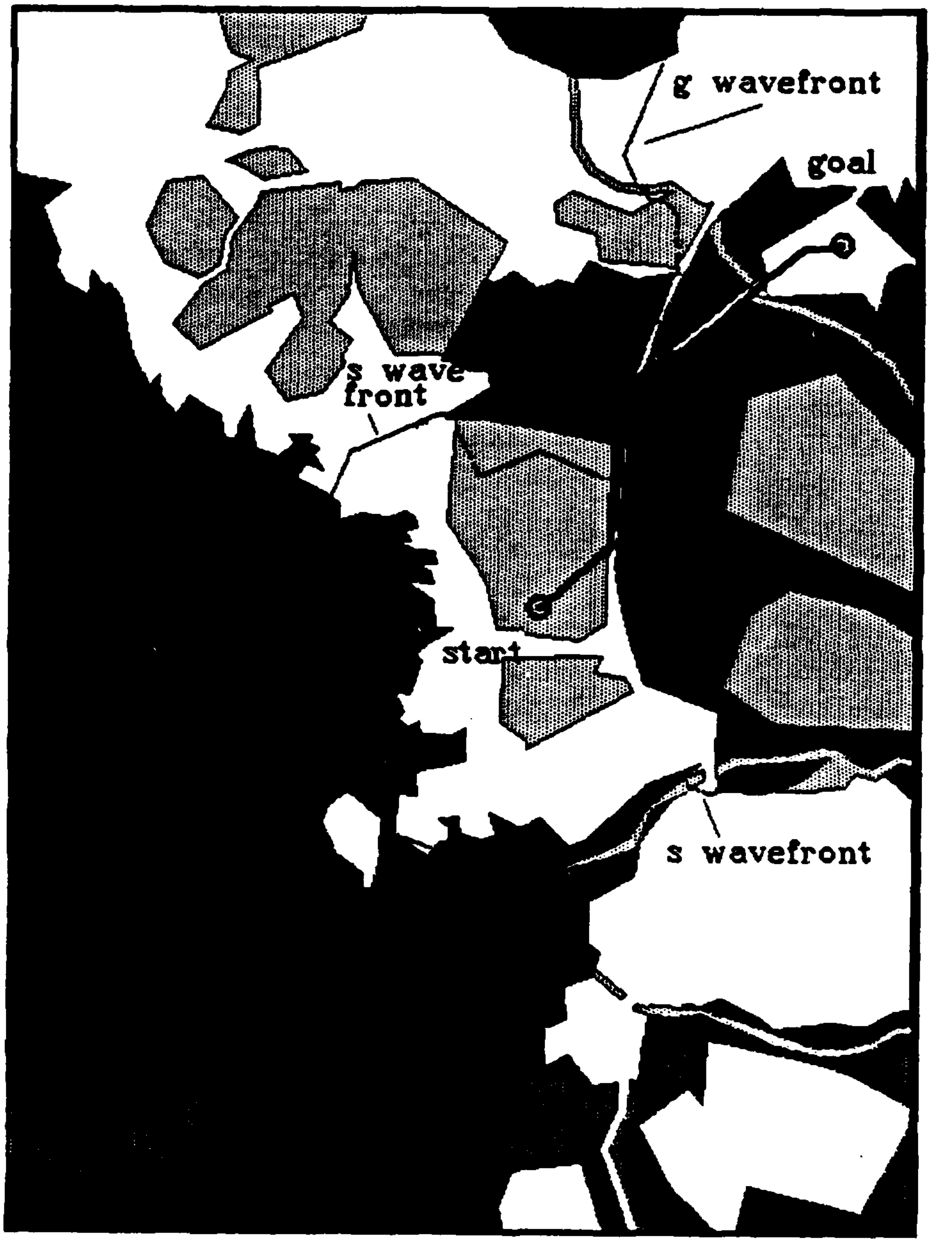


Figure 39. Problem E, Bidirectional Strategy

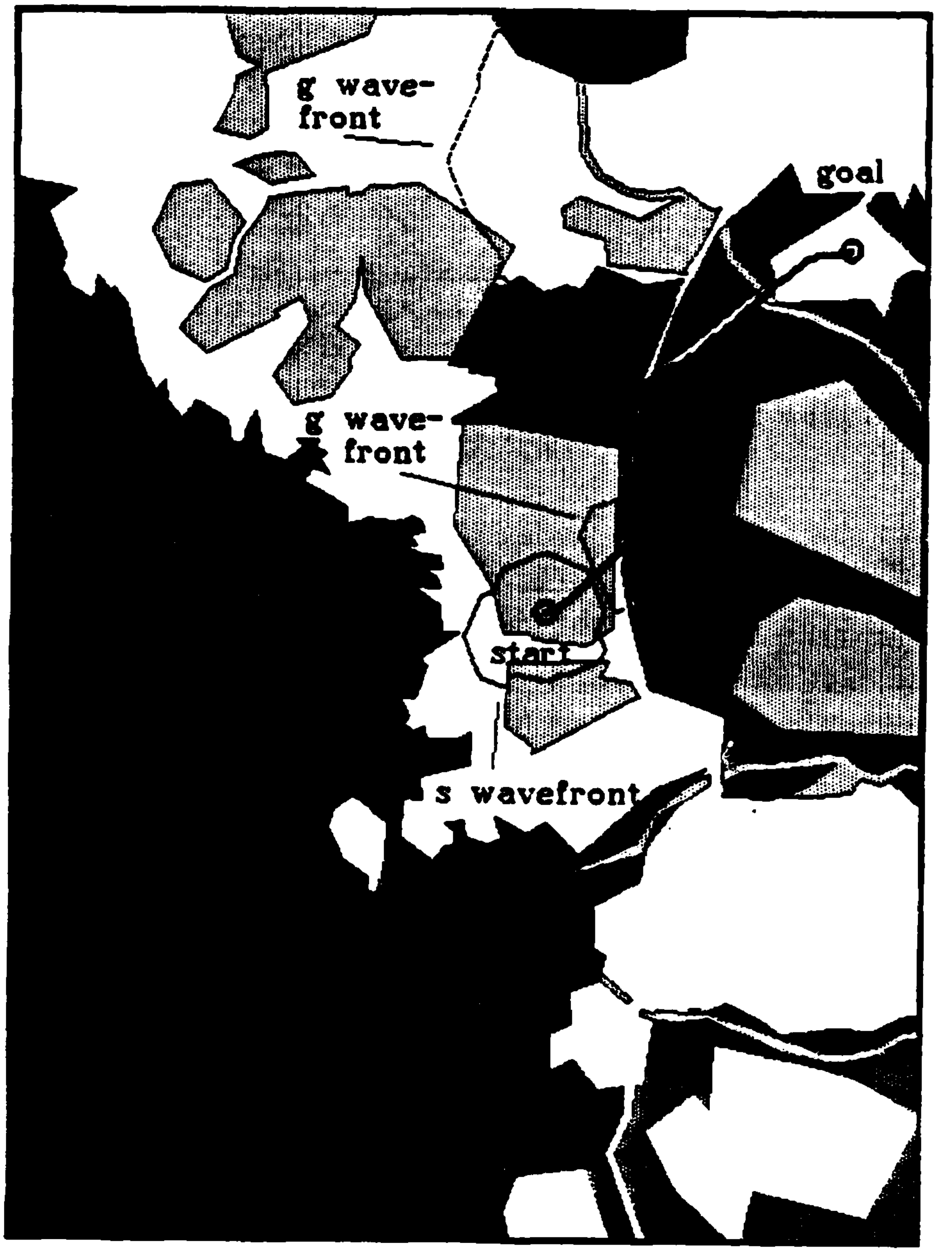


Figure 40. Problem E, Heuristic-Selection Strategy

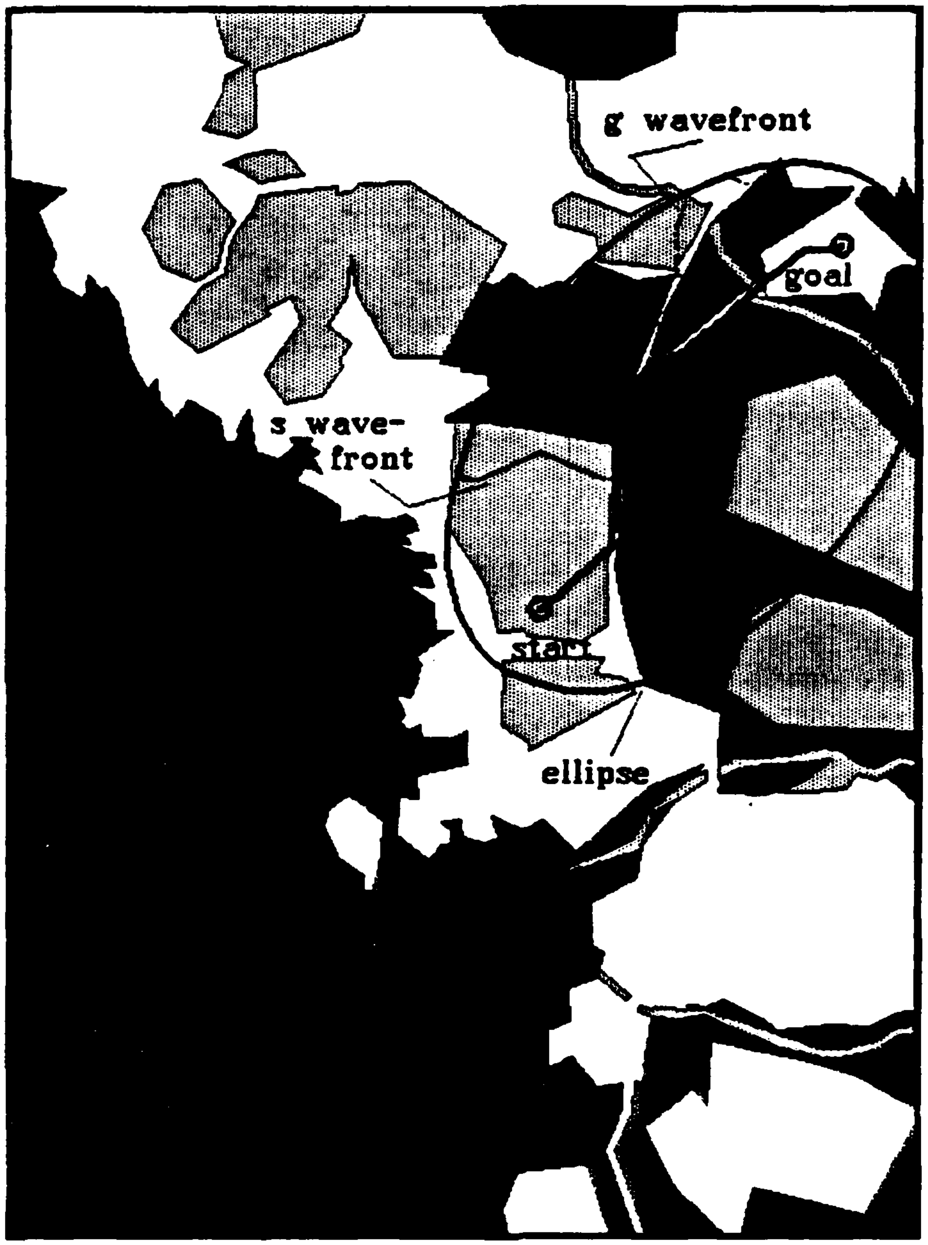


Figure 41. Problem E, Ellipse Strategy

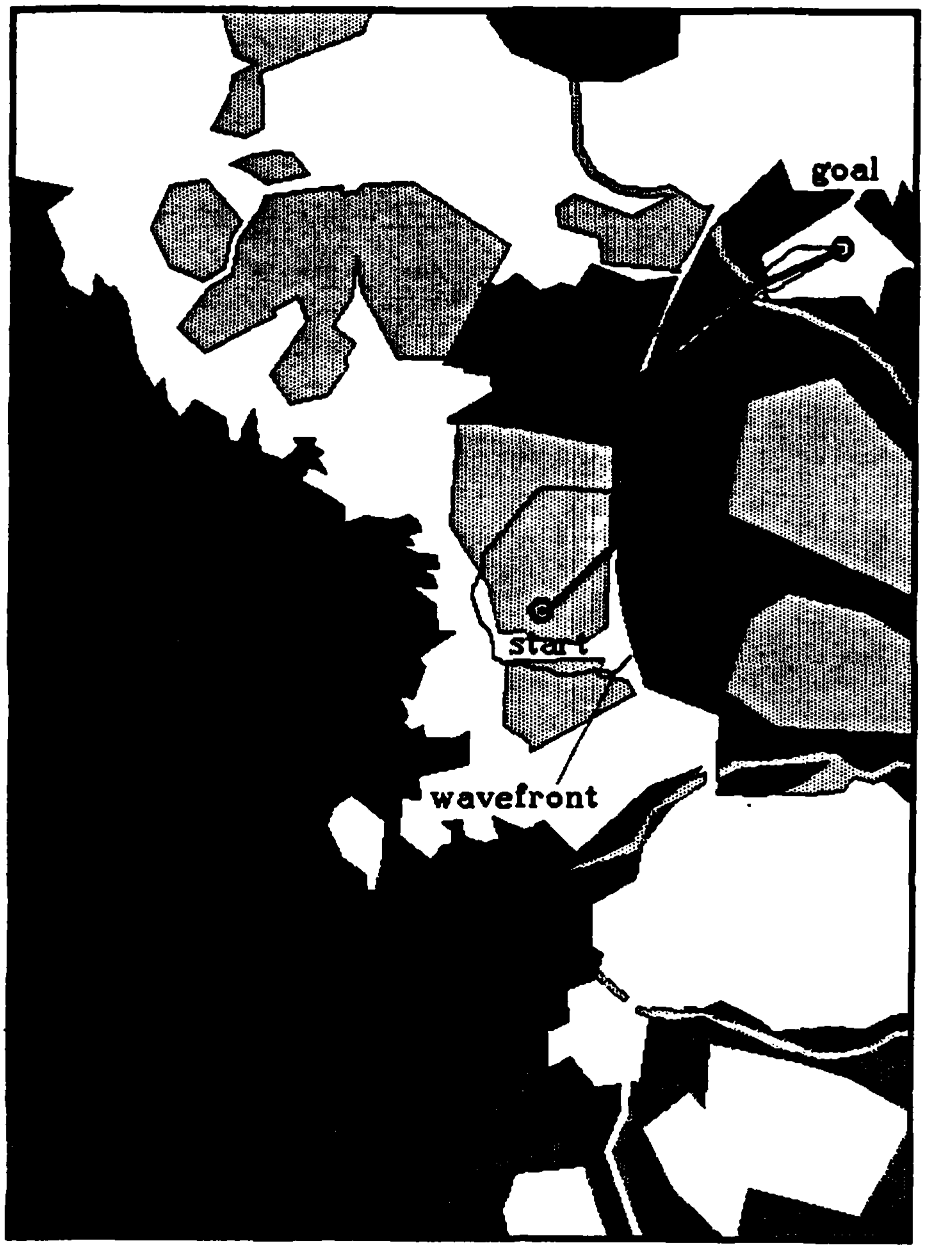


Figure 42. Problem E, A* Strategy

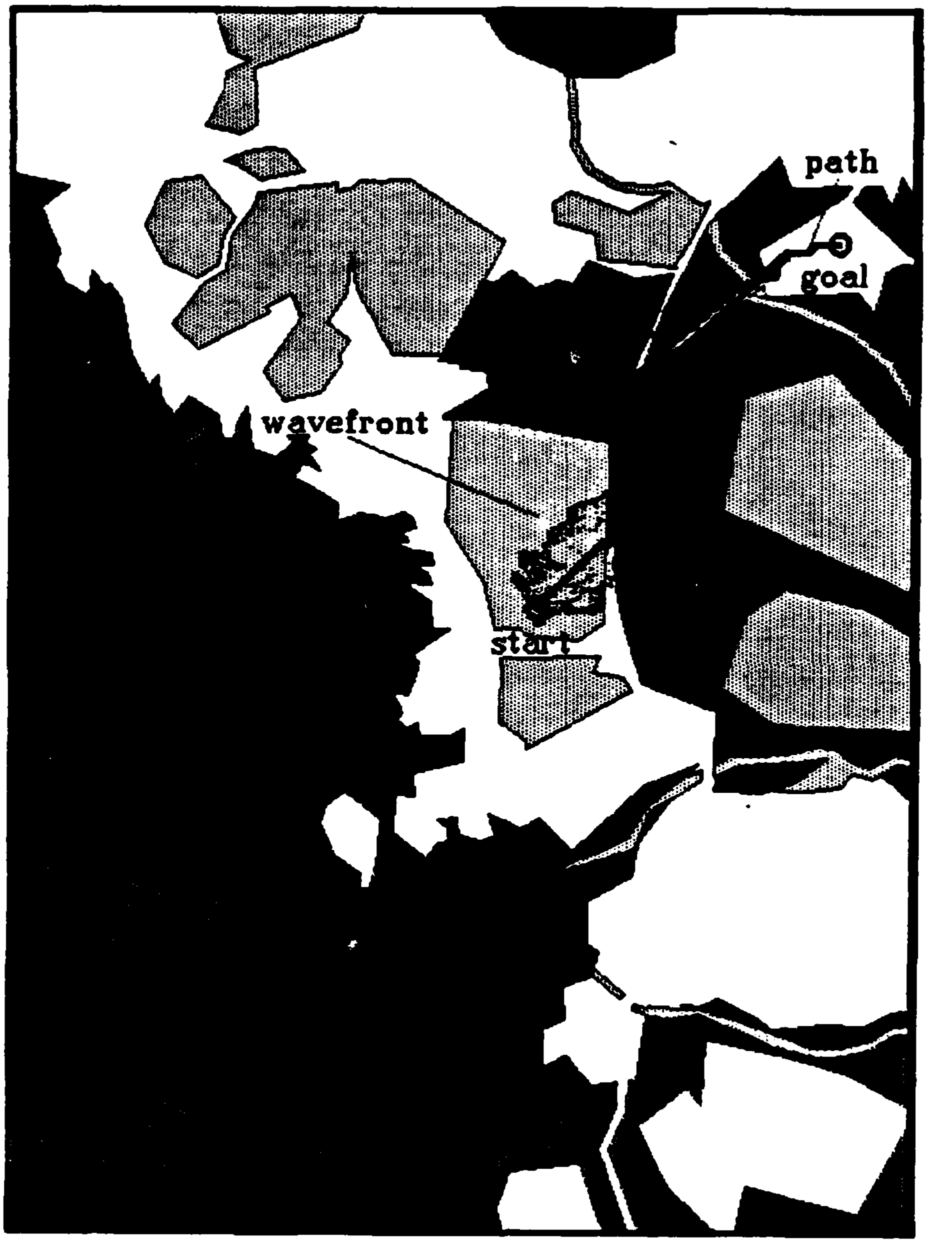


Figure 43. Problem E. A_ϵ* Strategy

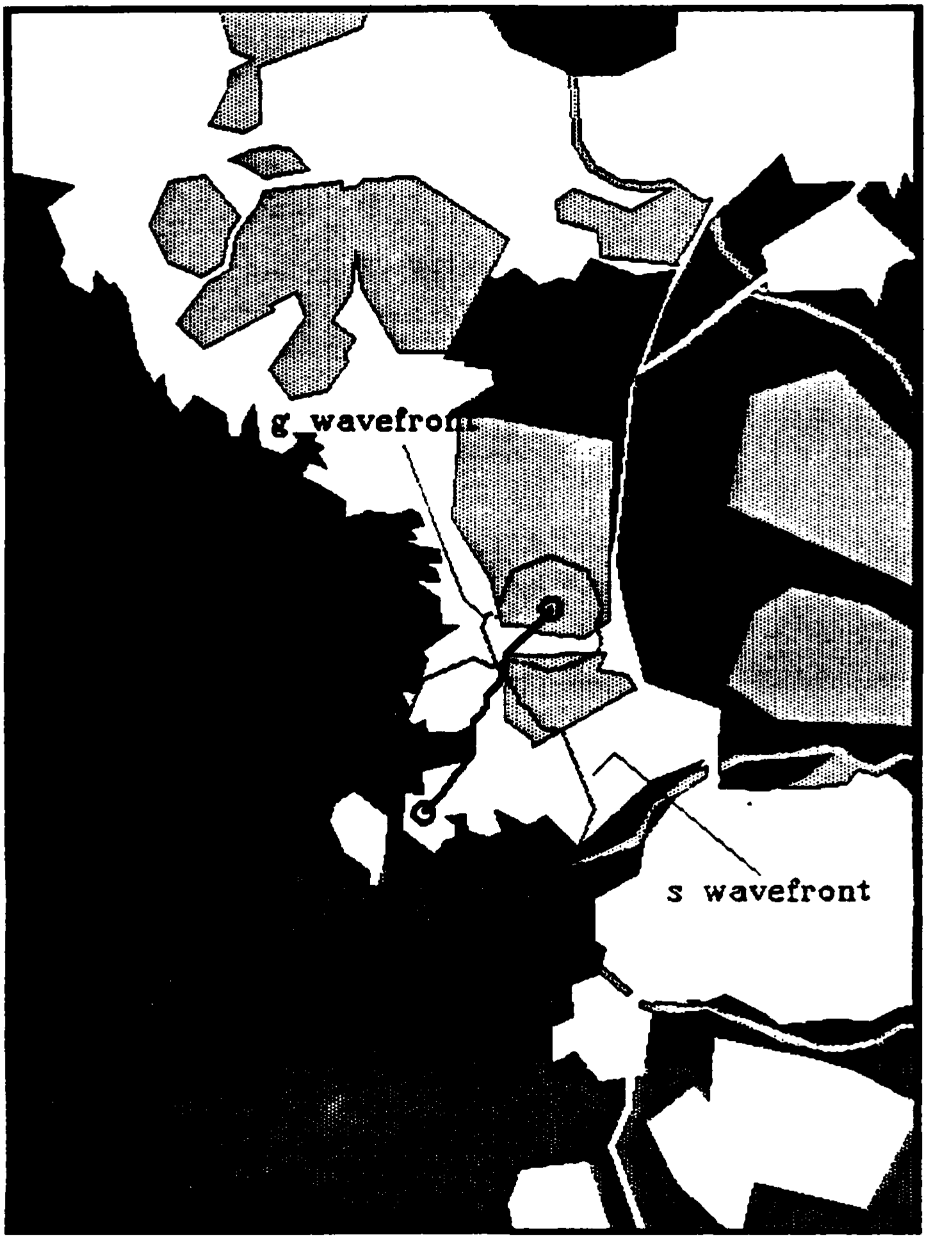


Figure 44. Problem F, Heuristic-Selection Strategy

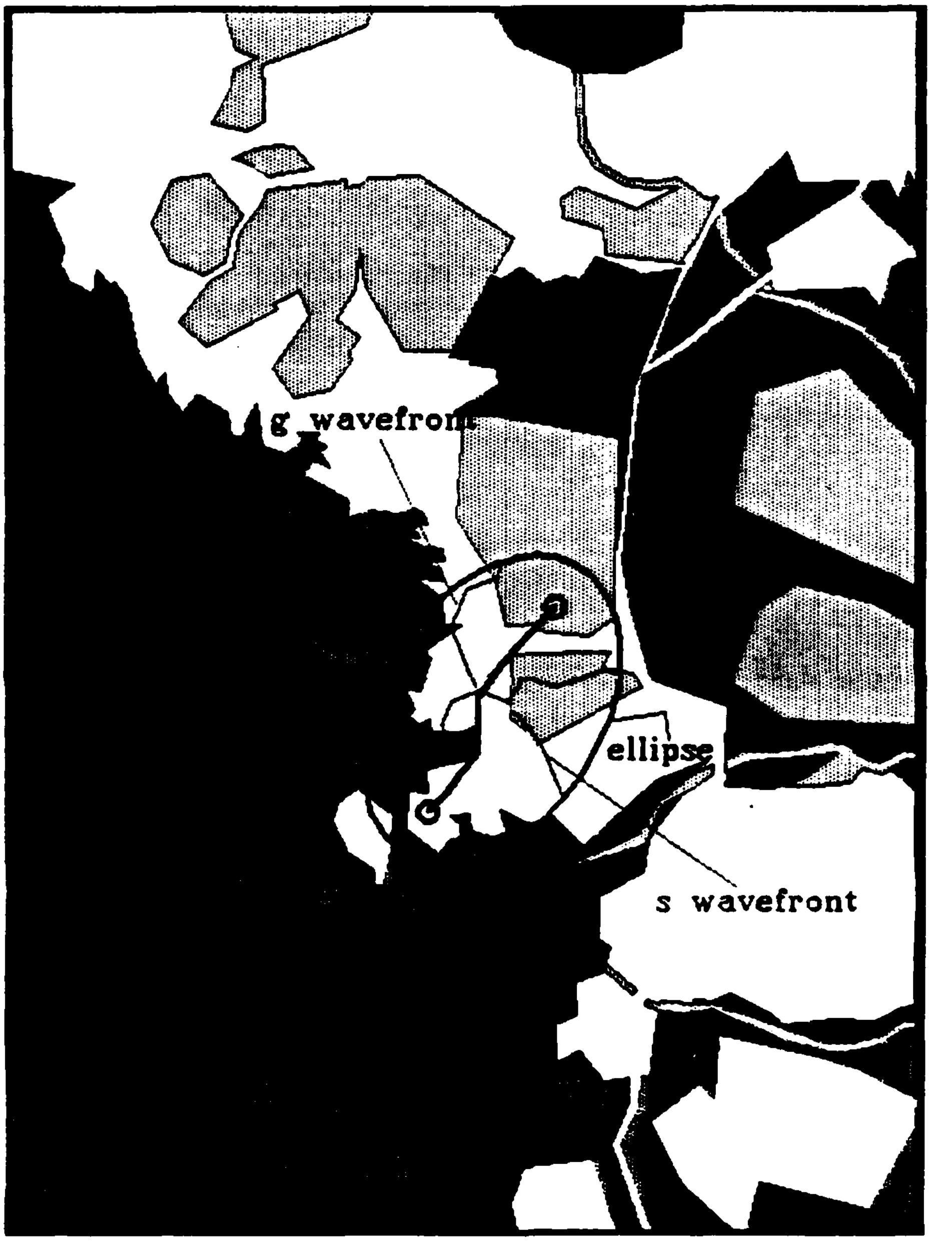


Figure 45. Problem F, Ellipse Strategy

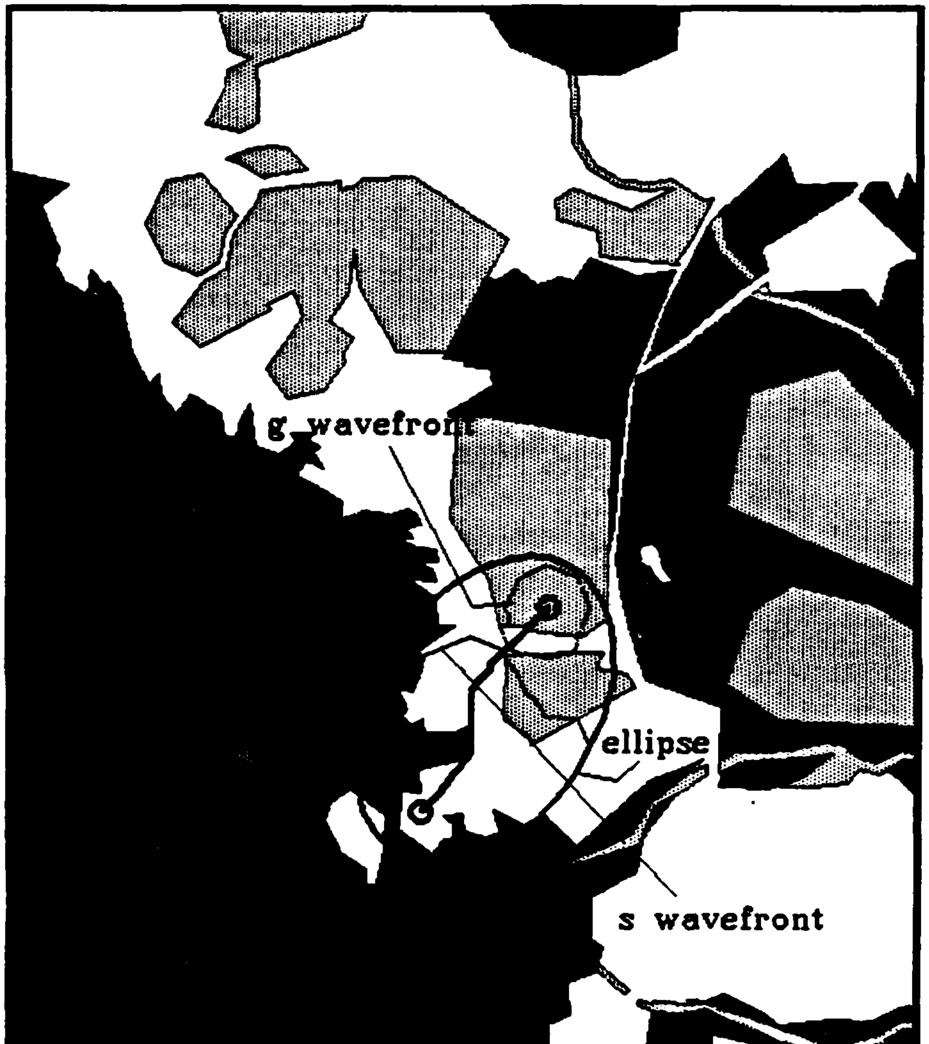


Figure 46. Problem F, Ellipse & Heuristic-Selection Strategy

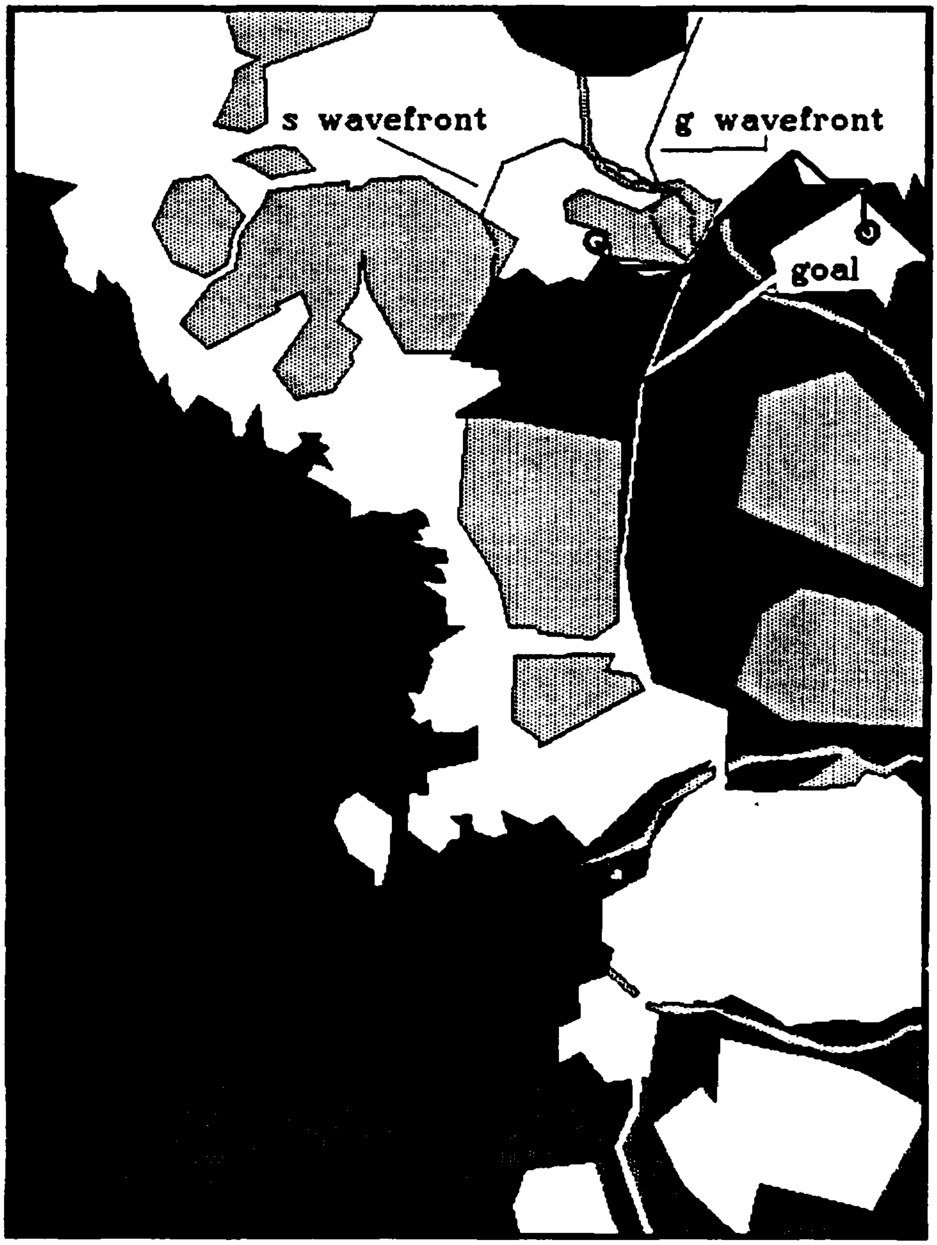


Figure 47. Problem G. Heuristic-Selection Strategy

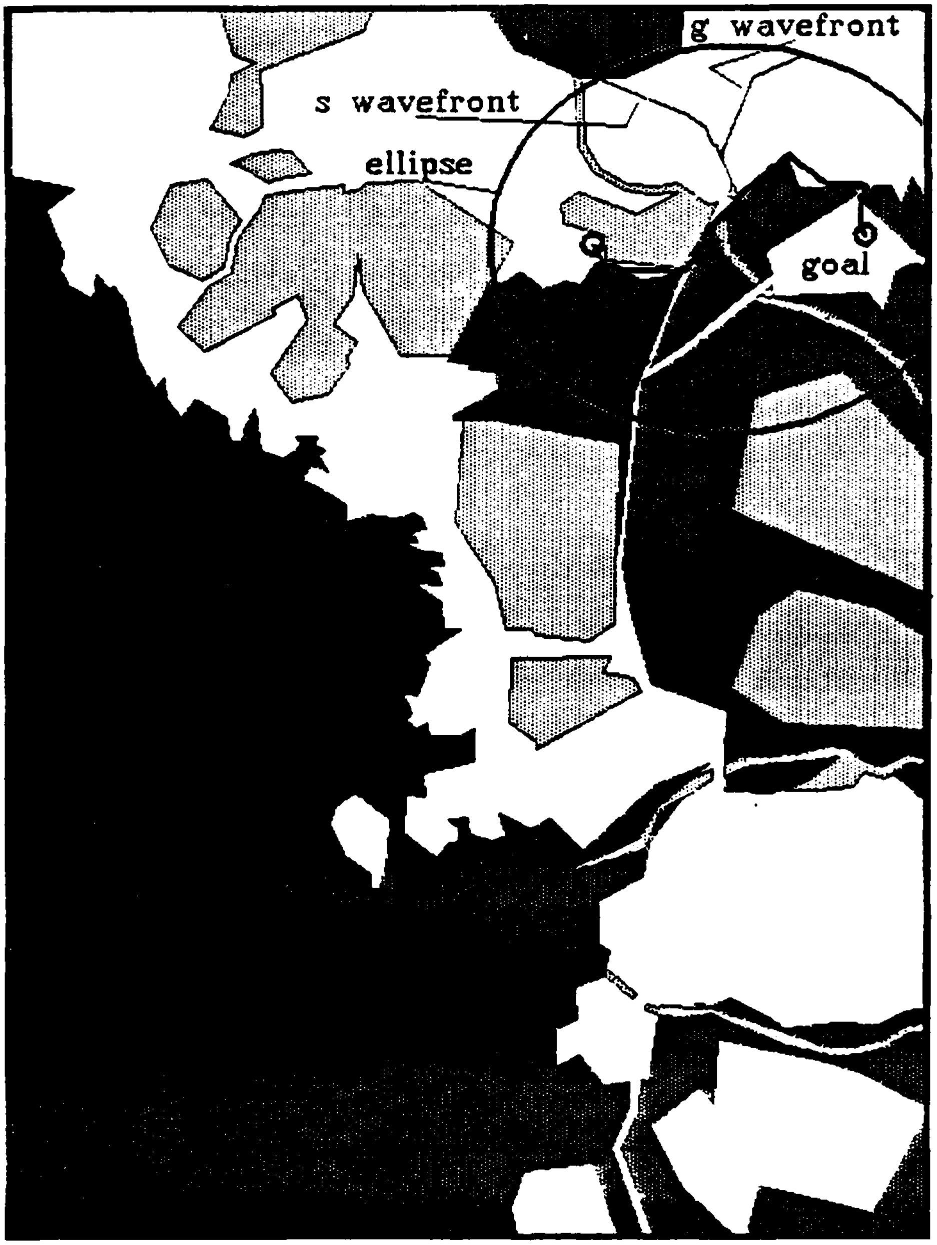


Figure 48. Problem G, Ellipse Strategy

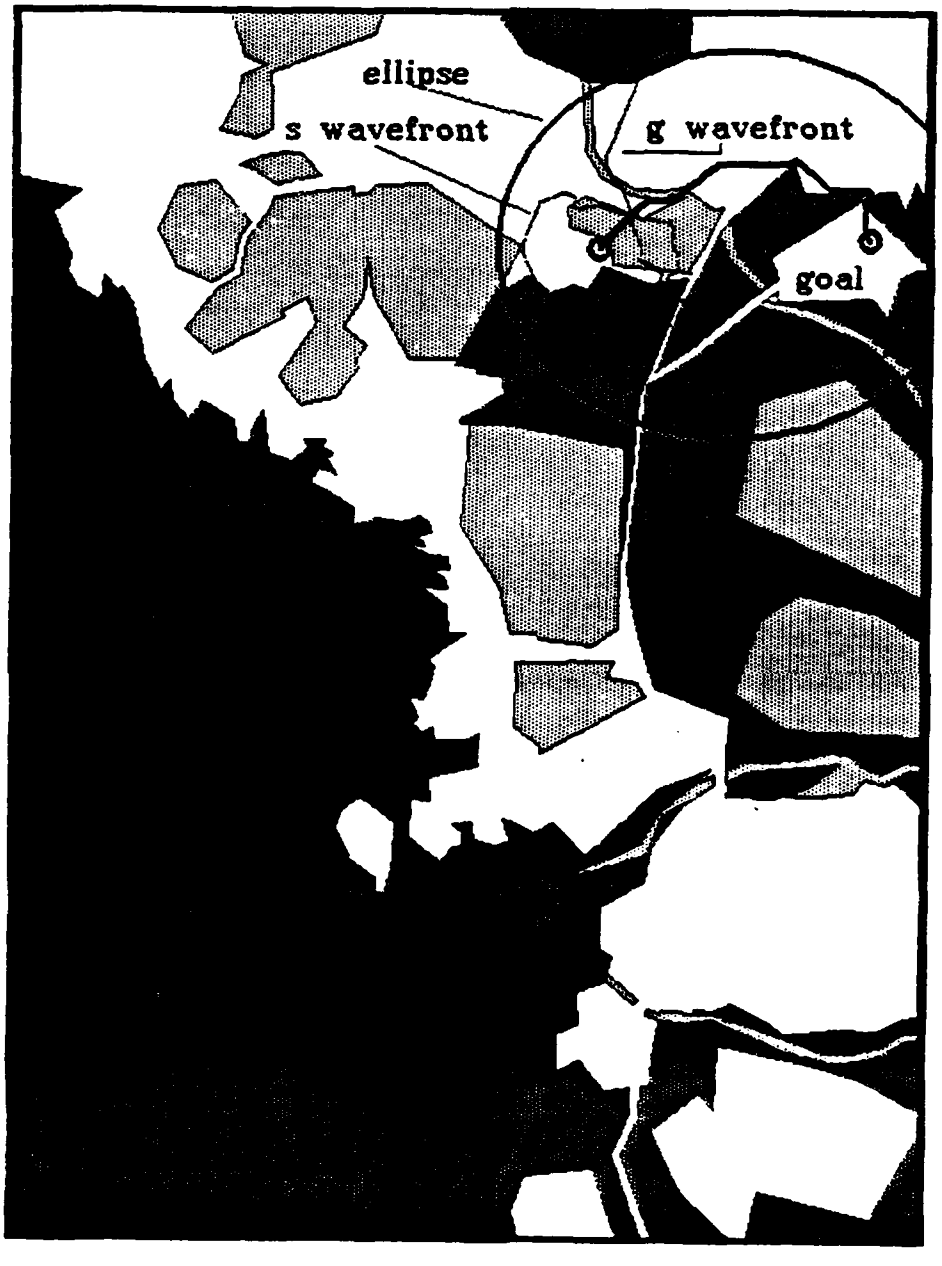


Figure 49. Problem G, Ellipse & Heuristic-Selection Strategy

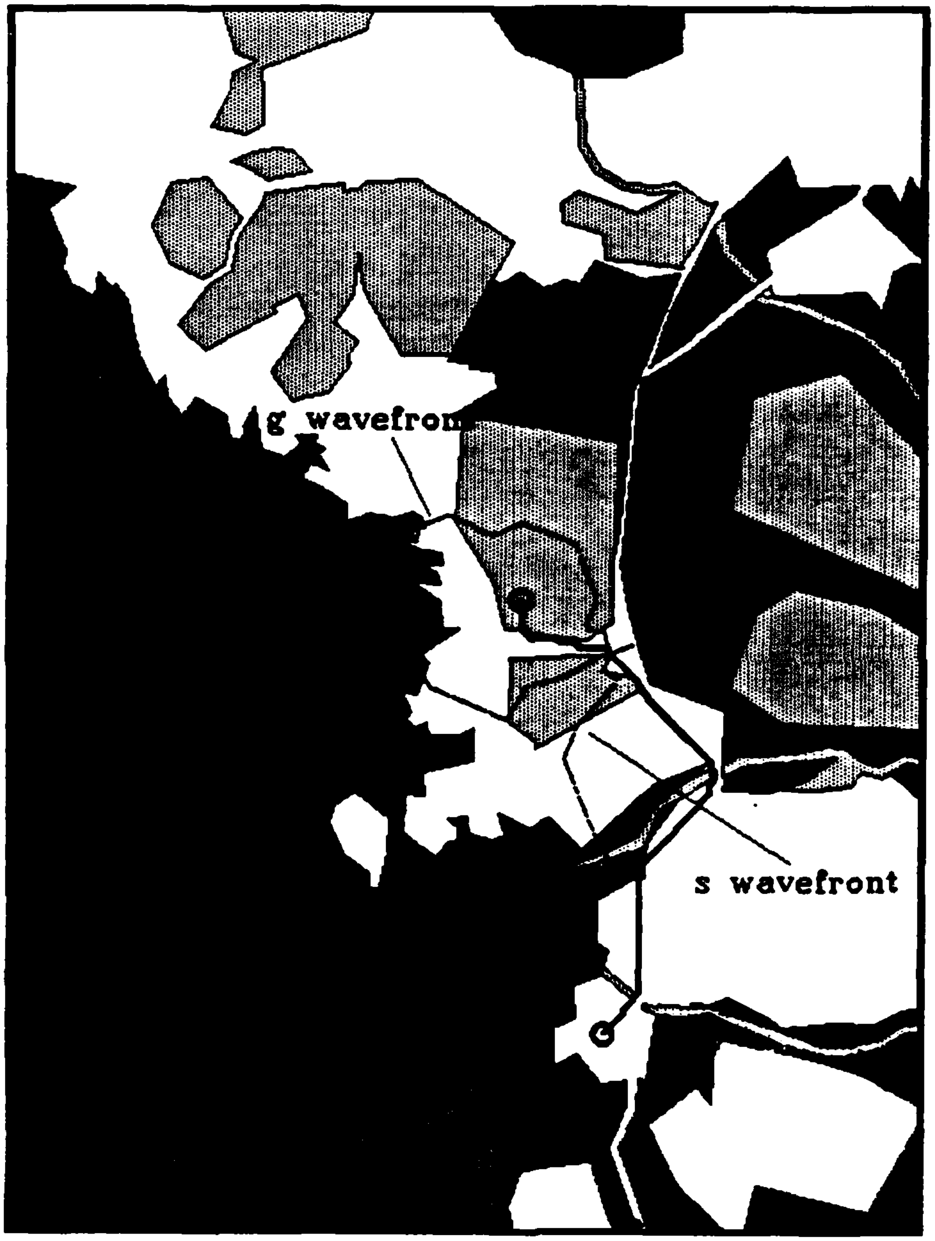


Figure 50. Problem H, Heuristic-Selection Strategy

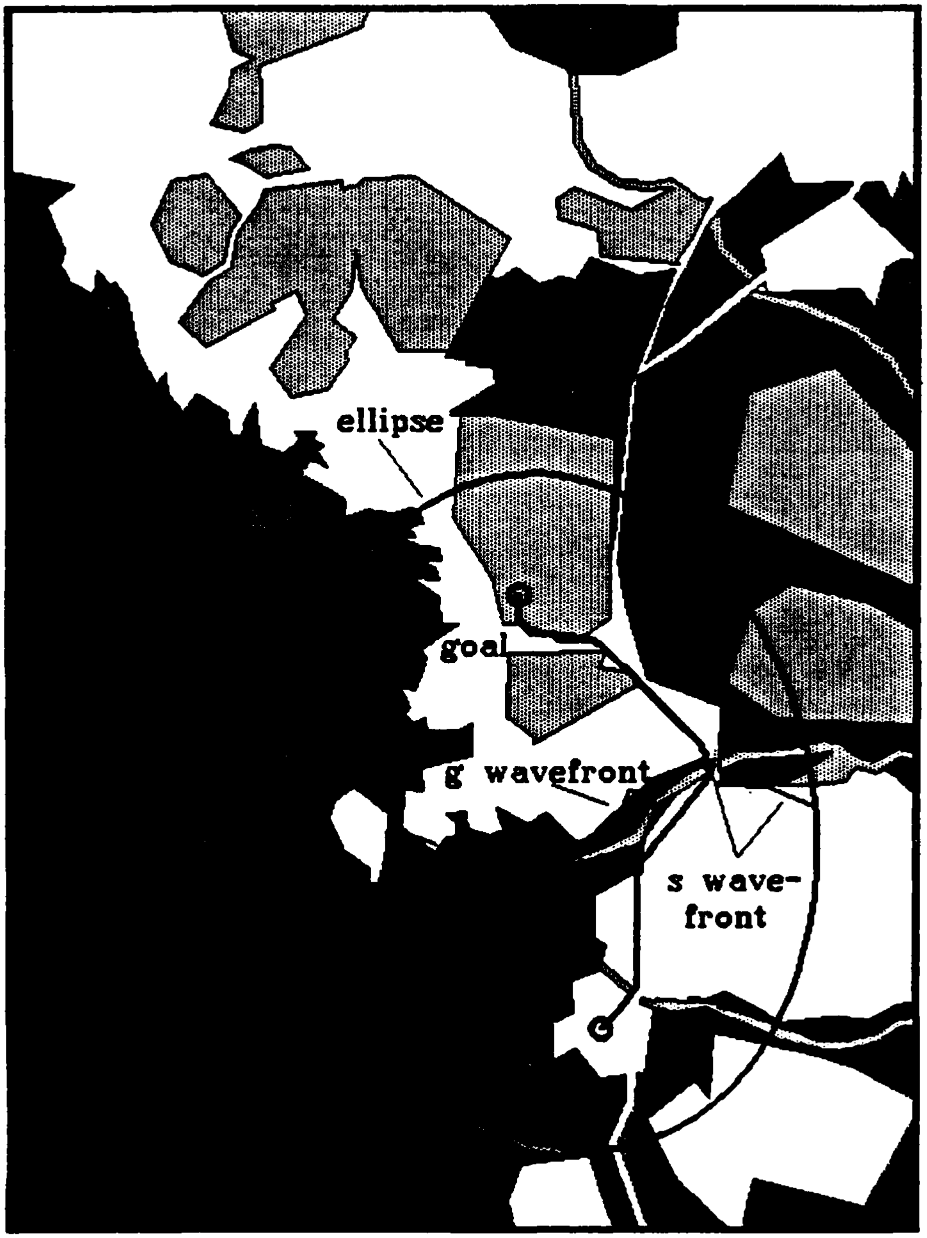


Figure 51. Problem H, Ellipse Strategy

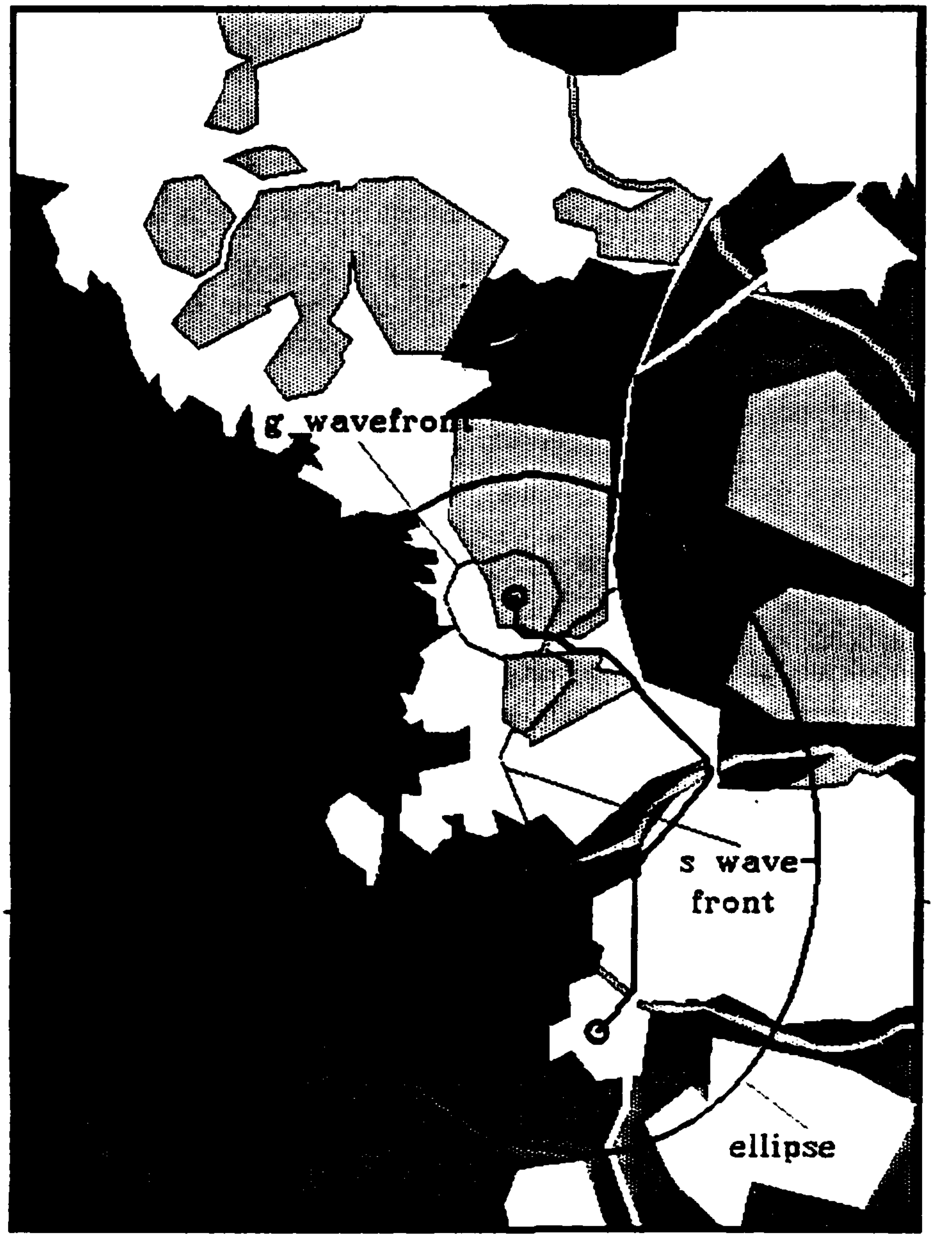


Figure 52. Problem H, Ellipse & Heuristic-Selection Strategy

IV. PROPERTIES OF SNELL'S-LAW PATHS

A. INTRODUCTION

The preceding chapter presents a classic conceptually-simple method of solving the weighted-region problem. This wavefront-propagation technique relies on a specialized artificial problem representation to overcome the weighted-region problem complexities. However, the simplifications made possible by the lattice-based problem representation have attendant drawbacks. As noted in Section III.A, altering the control-flow scheme for the algorithm cannot overcome the difficulties associated with the lattice-based problem representation. Thus, the creation of a more intelligent problem representation is desirable.

The homogeneous-cost region representation does not rely on a resolution-dependent lattice to describe the area-cost map. Areas in the environment that have the same cost-rate characteristics are viewed as single entities, not as a set of discrete points. The only points that are specifically used in the representation are those required to define region boundaries, i.e., the region vertices. Resolution can be changed by modeling the region polygons with greater or lesser precision. The disadvantage of relying on a homogeneous-cost region problem representation is that a simple graph search strategy may no longer applicable. A more complex algorithm, such as the continuous Dijkstra technique, could be required. We have seen that Snell's law provides a suitable basis for such an algorithm. However. some care is necessary when applying Snell's law to the weighted-region problem.

In this chapter, we first develop the basic terminology and definitions necessary to discuss the application of Snell's law to the weighted-region problem. Next, the applicability of the law itself is formally established. We prove useful properties of Snell's-law paths and develop characteristics of physically adjacent pairs of Snell's-law paths. We present Snell's-law-based factors that serve to constrain the search space for the weighted-region problem. Finally, we discuss limitations in applying the law to the weighted-region problem.

B. TERMINOLOGY AND DEFINITIONS

This section presents basic terminology and definitions. Some terms have already been referenced in preceding discussions. In those instances, the terms denoted their intuitive meanings. We now present formal definitions that hold for the remainder of this work. Figure 53 illustrates most definitions.

Agent: An agent is an entity capable traveling along the paths that represent solutions to instances of the weighted-region problem.

Cost Rate: A cost rate is a cost per unit of distance traveled along a path. In this thesis we consider cost rates to be independent of the direction of travel and independent of the time that travel occurs. A cost rate is defined based on the capabilities of a specific agent.

Homogeneous-Cost Region: A homogeneous-cost region is an polygonal area where the cost rate is the same everywhere within the polygon. Each polygon side demarcates a cost-change boundary so that the cost rate is different on either side of each boundary (but always the same inside the polygon). All contiguous areas

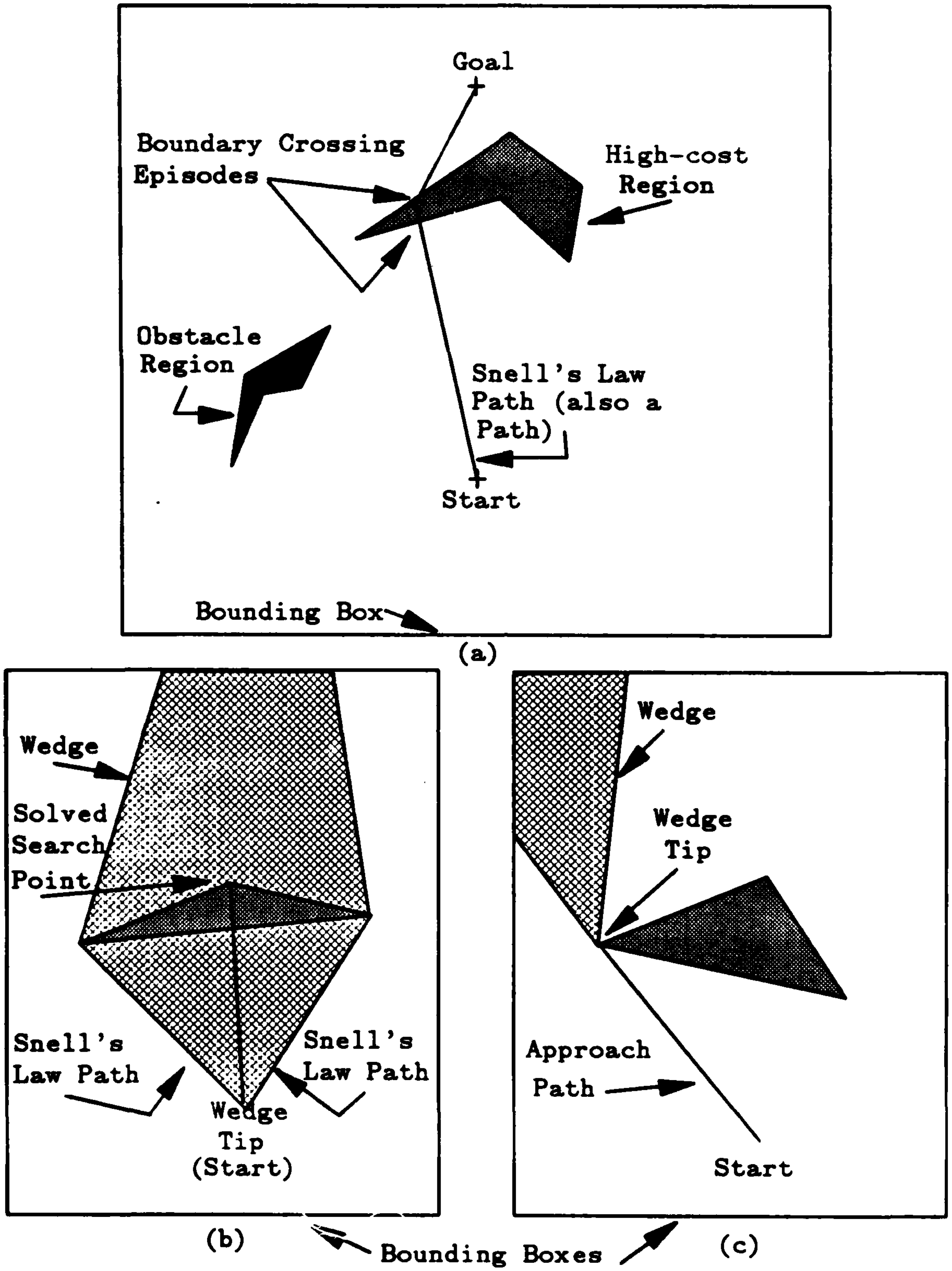


Figure 53. Definitions Illustrations

that have the same cost rate are grouped together into a single homogeneous-cost region. Thus, the polygons delimiting the homogeneous-cost regions may be either convex or non-convex.

Obstacle: An obstacle is a polygonal area that cannot be traversed. Conceptually, an obstacle is a homogeneous-cost region that has an infinite cost rate.

Area-Cost Map: An area-cost map is a planar thematic map (where the theme of the map is cost rates) representing some physical area such that the area is described by disjoint homogeneous-cost region polygons and obstacle-area polygons. The area-cost map associates a specific cost rate with each homogeneous-cost region.

Ternary-Cost Map: A ternary-cost map is an area-cost map that recognizes only three distinct cost rates (assigned to any number of regions). The different cost rates are characterized as infinite, high and low. Obstacle areas have conceptually infinite cost rates. Polygons are used to delimit high-cost homogeneous-cost regions. Conceptually, the high-cost homogeneous-cost regions and obstacle regions are superimposed on a "background" area. The background areas are traversable at low (or equivalently, optimal) cost and are referred to as low-cost regions or optimal-cost regions.

Start: The start is a point in the two-dimensional plane that specifies the initial position of an agent.

Goal: The goal is a point in the two-dimensional plane that specifies the desired (terminal) location of an agent.

Path: A path is a connected ordered series of line segments (or equivalently, path segments) that begins at a specific starting point and ends at a specific goal point.

Boundary-Crossing Episode: A boundary-crossing episode occurs when a path intersects a side of a homogeneous-cost region polygon. The episode occurs at the point of intersection. This intersection point is treated as the common endpoint for two consecutive path segments on the path. Note that some path segments may have a region vertex as an endpoint while not intersecting a side of a homogeneous-cost region at that vertex. Such paths can occur for example, when a path goes around a high-cost region. In these cases, the path does not have a boundary-crossing episode at the region vertex.

Weighted-Region Problem: The weighted-region problem is the problem of locating the minimum-cost path between a start and goal, given an area-cost map that includes those two points and the optimal-cost path between them. Specifically, let P_{SG} be the set of all start-to-goal paths that obey Snell's law at each boundary-crossing episode on the path such that $p_i \in P_{SG}$. Assume each p_i includes n_i path segments and that a unique cost rate c_j can be associated with each path segment. Let the distance along each path segment of p_i be d_j. (Note that $j \in [1..n_i]$ for each p_i). Then the weighted-region problem that we solve is a minimization problem:

$$\min_{p_i \in P_{SG}} \sum_{j=1}^{j=n_i} c_j d_j$$

Snell's-Law Path: A Snell's-law path is a path such that Snell's law is obeyed at each boundary-crossing episode that occurs on the path. Consecutive path segments that do not correspond to boundary-crossing episodes are not required to obey Snell's law.

Feasible Path: A feasible path is a path that does not intersect any obstacle region.

Cost-Limiting Path: A cost-limiting path is a feasible start-to-goal path of computable cost. This path provides an upper bound (limit) on the cost of the optimal start-to-goal path.

Bounding Box: A bounding box is a rectangle that delineates the portion of an area-cost map that must contain the optimal-cost solution path to an instance of the weighted-region problem. The size of a bounding box is usually determined by a cost-limiting path.

Search Point: A search point is a point in two-dimensional space that corresponds to the goal, a vertex of a homogeneous-cost region polygon, or a vertex of an obstacle-area polygon.

Wedge: A wedge is a portion of the area-cost map that is defined by two Snell's-law paths having the same starting point. One Snell's-law path defines the left side of the wedge, the other path defines the wedge's right boundary (left and right are defined from the point of view of an observer positioned at the common starting point and looking towards the interior of the wedge). A wedge ends. or terminates. when the two boundary-defining Snell's-law paths intersect or intersect the bounding box. Thus, a wedge can be described as a polygon that has two Snell's-law paths and, possibly, a portion of the bounding box, as sides.

Search Point Within A Wedge: A search point is within a wedge if that point is interior to the polygon describing the wedge. Search points that are vertices of the wedge-describing polygon are not considered to be within the wedge.

Snell's-law Path Within A Wedge: A Snell's-law path is within a wedge if it contains a point within the wedge and it does not pass through any side of the polygon describing the wedge. A Snell's-law path may touch a wedge-describing polygon side without passing through that side.

Wedge Tip: The common starting point of the two Snell's-law paths defining a wedge is the wedge tip.

Solved Search Point: A search point is solved with respect to a given wedge if the search point is interior to the polygon describing the given wedge and there is a Snell's-law path, also entirely within the same wedge-describing polygon, from the wedge tip to that search point.

Unsolved Search Point: A search point is unsolved with respect to a specific wedge if the search point is inside the wedge-describing polygon and no Snell's-law path, entirely within the wedge, has been found from the wedge tip to the search point.

Empty Wedge: When the polygon delimiting a wedge contains no interior unsolved search points, the wedge is empty.

Well-Behaved Snell's-Law Path Pair (WBSP): A well-behaved Snell's-law path pair is defined by two Snell's-law paths. The two paths are well-behaved with respect to each other if they each intersect the same sequence of homogeneous-cost region boundaries in the same order.

Partial Well-Behaved Snell's-Law Path Pair (K-WBSP): Two Snell's-law paths form a K-WBSP when they are well-behaved with respect to each other only up through their first K boundary-crossing episodes. Note that any two Snell's-law paths are trivially 0-WBSP when they share the same starting point and have no other boundary-crossing episodes in common. Note that we differentiate between WBSP and K-WBSP. If two Snell's-law paths form a WBSP then they do not, by definition, form a K-WBSP for any K. (In Figure 53(b), the 2 Snell's-law paths are 2-WBSP.)

Explored Wedge: An explored wedge is an empty wedge defined by two Snell's-law paths that form a WBSP.

K-Explored Wedge: A K-explored wedge occurs when the two Snell's-law paths defining the wedge form a K-WBSP and the wedge contains no unsolved search points up to the Kth boundary. Then any new Snell's-law path constructed within the wedge will form an N-WBSP with each of the extant wedge-defining Snell's-law paths, where $N \geqslant K$. Note that any two Snell's-law paths having the same starting point trivially define a 0-explored wedge.

Closest Unsolved Search Point: Each non-empty wedge contains one or more unsolved search points. Each wedge must be K-explored (for $K \geqslant 0$). The closest unsolved search point is the unsolved search point that is closest to the Kth boundary (or point in some cases such as $K = 0$).

Approach Path: When the wedge tip is not the start, there must be some known approach path, leading from the start to the wedge tip. Any Snell's-law path within the wedge must include the approach path as its initial portion. An approach path is defined relative to a specific wedge.

C. DERIVATION AND STATEMENT OF SNELL'S LAW

Consider the problem of finding the optimal-cost start-to-goal path in a simple case. Figure 54 depicts a situation where the start is located in a low-cost homogeneous-cost region, the goal is inside a high-cost homogeneous-cost region and a single region boundary, denoted boundary B, lies between them. Thus, in Figure 54, U1 is a lower cost rate than U2. Let the series of path segments from the start to point P to the goal represent the optimal-cost start-to-goal path. This least-cost path is a perturbation of the straight-line start-to-goal path that trades increased path length in the low-cost (U1) region for decreased path length in the high-cost (U2) region. To find the optimal-cost start-to-goal path, we must find the point P on boundary B that minimizes path cost by maximizing the advantage of the tradeoff.

We can write an equation that expresses the cost of the two-path-segment path in Figure 54. Let C denote path cost. Then, the equation describing C for Figure 54 is a sum of terms as below:

$$C = \frac{\left(x1^2 + y1^2 \right)^{1/2}}{U1} + \frac{\left(x2^2 + y2^2 \right)^{1/2}}{U2}$$

Note that, as shown in Figure 54, y1 and y2 are constants. Thus, we can take partial derivatives of C with respect to $x1$ and $x2$ and set them equal to zero to characterize a minimum-cost path between the start and goal.

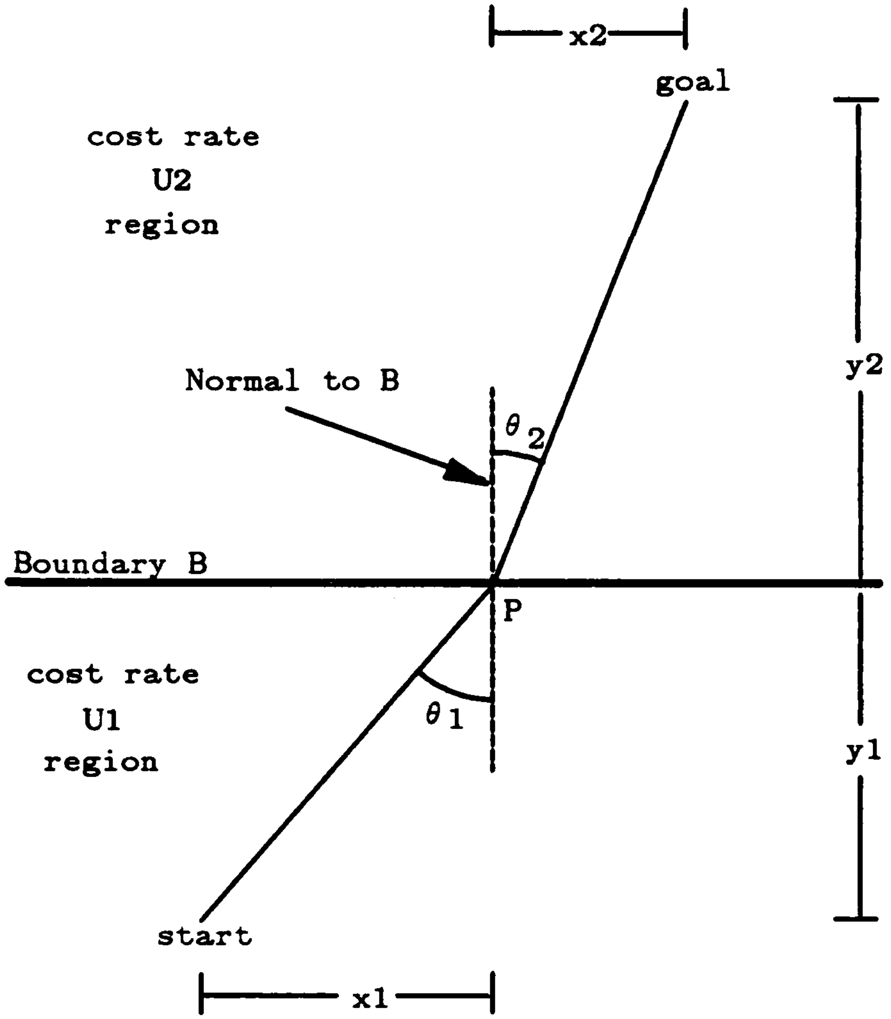


Figure 54. Minimal-Cost Path Derivation

$$\frac{\partial C}{\partial x1} = \frac{x1}{U1\left(x1^2 + y1^2\right)^{1/2}}$$

$$\frac{\partial C}{\partial x2} = \frac{x2}{U2\left(x2^2 + y2^2\right)^{1/2}}$$

Since both partial derivatives are set equal to zero, we can write:

$$\frac{x1}{U1\left(x1^2 + y1^2\right)^{1/2}} = \frac{x2}{U2\left(x2^2 + y2^2\right)^{1/2}} \qquad (1)$$

Note that the start-to-goal path can also be characterized in terms of the angles that it makes with a normal to boundary B through point P. In Figure 54, these angles are denoted θ_1 and θ_2. Since the sine of an angle in a right triangle is equal to the length of the side opposite the angle divided by the length of the triangle hypotenuse, we can express the sine values of θ_1 and θ_2 as:

$$\sin(\theta_1) = \frac{x1}{\left(x1^2 + y1^2\right)^{1/2}}$$

$$\sin(\theta_2) = \frac{x2}{\left(x2^2 + y2^2\right)^{1/2}}$$

These sine values can be substituted into equation (1), resulting in a simplified equation that describes the minimal-cost start-to-goal path:

$$\frac{\sin(\theta_1)}{U1} = \frac{\sin(\theta_2)}{U2}$$

This final equation is exactly Snell's law [Ref. 40, p. 147]. We note that the relation expressed by Snell's law is entirely a local relation. It is easily shown. by induction, that a minimum-cost path between two points that involves an arbitrary number of boundary-crossing episodes must obey Snell's law (locally) at each boundary-crossing episode. Thus, Snell's law is a local optimality criterion for solving the weighted-region problem. (Also see [Ref. 3, p.10] for a similar proof.) Figure 55 illustrates Snell's law.

D. CONVEXITY OF THE SNELL'S-LAW PROBLEM

We have demonstrated that applying Snell's law when moving across polygon boundaries defining different cost regions allows the solution of a minimization problem involving Euclidean distances divided by the appropriate cost coefficients. (Note that we are using cost reciprocals, an arbitrary decision.) We now show that this minimization problem is convex. Figure 56 depicts a typical instance of the problem that we wish to solve. Formally stated, the problem is:

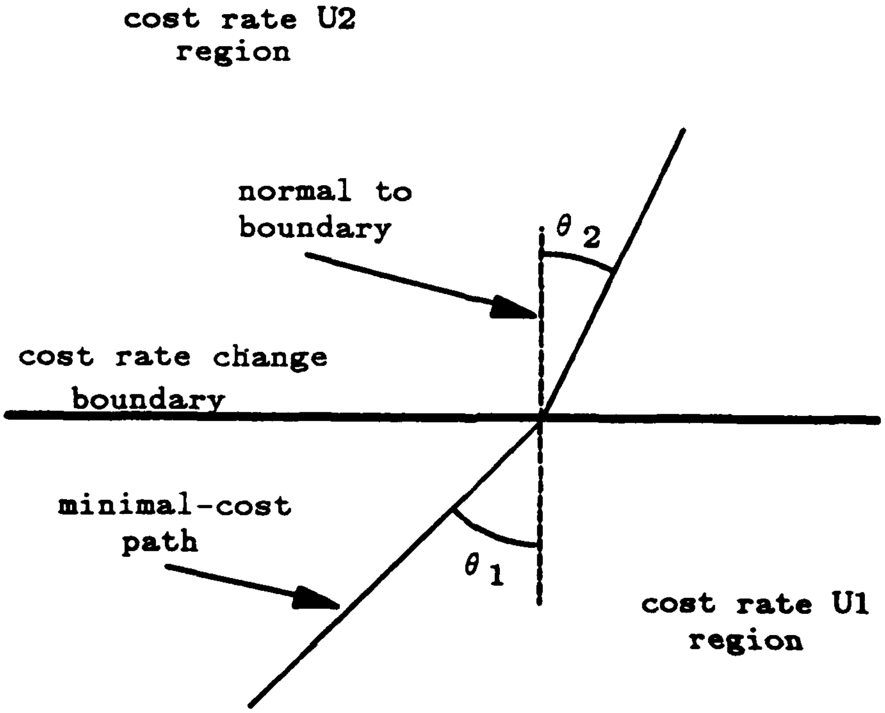
$$\min \frac{\left((x_2-x_1)^2+(y_2-y_1)^2\right)^{1/2}}{c_1}+\frac{\left((x_3-x_2)^2+(y_3-y_2)^2\right)^{1/2}}{c_2}+\frac{\left((x_4-x_3)^2+(y_4-y_3)^2\right)^{1/2}}{c_1}$$

Subject To:

$$A_1x_2+B_1y_2+C_1=0$$
$$A_2x_3+B_2y_3+C_2=0$$

where $A_1x+B_1y+C_1=0$ is the equation of the lower region boundary and $A_2x+B_2y+C_2=0$ is the equation of the upper region boundary. The variables are x_2, x_3, y_2 and y_3. The coordinates x_1, y_1, x_4 and y_4 are constant, the coordinates of the start and goal location.

First, consider the convexity of the Euclidean-distance problem involving only a fixed point (such as the start or goal), a point on a region boundary (i.e., a line) and a single cost rate, c. (The variables for this problem are x_2 and y_2, the



$$\sin(\theta_1)/U1 = \sin(\theta_2)/U2$$

Figure 55. Snell's Law

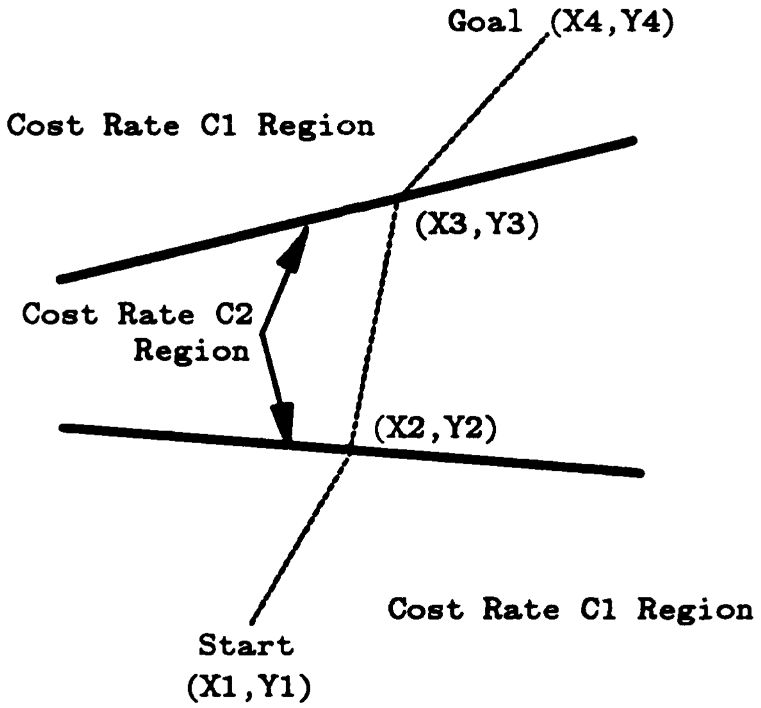


Figure 56. Convexity Illustration

coordinates for the point on the region boundary.) This problem is:

$$\min \frac{\left((x_1-x_2)^2+(y_1-y_2)^2 \right)^{1/2}}{c}$$

Subject To:

$$A_1x_2+B_1y_2+C_1=0$$

Note that Euclidean distances are always positive. Therefore, the problem is restricted to the positive quadrant. In this case, minimizing the square of the function is an equivalent problem. Thus, the problem objective function can be simplified to:

$$\min \frac{(x_2-x_1)^2+(y_2-y_1)^2}{c}$$

Note that the function $f(x_1,x_2)=(x_1-x_2)^2$ maps the reals to the positive reals. Therefore, we consider this function as equivalent to $g(X)=X^2$ where $X=x_1-x_2$. It is well known that $g(X) = X^2$ is a convex function (in terms of X, see [Ref. 13] for example). Therefore, $f(X)=\frac{X^2}{c}$ is convex. Therefore, $f(x,y)=\frac{(x-y)^2}{c}$ is convex.

Because the addition of two convex functions produces another convex function, $\frac{(x_2-x_1)^2+(y_2-y_1)^2}{c}$ is convex. Therefore, the Euclidean-distance function for our simplified problem is convex. The original objective function, as depicted in Figure 56, is also the sum of convex functions. Therefore, this objective function is convex. The constraints for the problem are linear by definition and are thus trivially convex (i.e.. both convex and concave). Therefore, the problem is the minimization of a convex objective function subject to convex constraints.

Adding one region boundary to the problem results in the objective function changing by the addition of a convex function and the constraints being augmented by another linear (convex) function. Therefore, the convex nature of the problem holds, regardless of the number of boundary-crossing episodes involved.

Convexity guarantees that any locally optimal-cost solution will have globally-optimal cost. However, if region boundaries are line segments instead of (infinite length) lines, the solution is only guaranteed to be the optimal-cost solution path among all paths that intersect the same boundary. Other paths, that go around the boundary for example, may have lower path cost. As the minimum-cost formulation above is equivalent to finding the Snell's-law path between two points, the latter problem shares these properties, including convexity (with respect to specific boundaries).

E. DEVELOPING PRELIMINARY RESULTS

We now develop several lemmas that characterize Snell's-law paths. The lemmas implicitly rely on the convexity of the problem. The first three lemmas all assume Snell's-law paths consisting of only two line segments and having one boundary-crossing episode. The results are later extended to arbitrary Snell's-law paths. For notation, let A-to-B denote a straight-line path from A to B.

Lemma 1: Given an initial Snell's-law path, R, that has a single boundary-crossing episode (over the entire length of the path) involving boundary B of a homogeneous-cost region at point P, then any other Snell's-law path, R_L, that also has only one boundary-crossing episode, intersecting boundary B at some point to the left of P, will lie entirely to the left of R (i.e., both below and above boundary B), and any Snell's-law path, R_R, that has a single boundary-crossing episode, intersecting boundary B at some point to the right of P, will lie entirely to the right of R (both below and above boundary B).

Proof: There are three cases.

Case 1: (See Figure 57) R intersects B so that its angle of incidence is normal to B. In this case, since R obeys Snell's law about boundary B, R is a straight line. No heading change along R occurs either before or after intersection with boundary B. Clearly, if some path R_L intersects B at a point to the left of P, then the angle of incidence between R_L and the normal to B must be measured in a counter-clockwise direction (again, as specified by Snell's law). According to Snell's law, the exit angle (i.e., the angle of refraction) of path R_L will also be measured in a counter-clockwise direction. Therefore, R_L always moves away from R to the left and must lie entirely to the left of R.

If some Snell's-law path R_R intersects B at a point to the right of P, similar reasoning holds, except that the angles are measured in a clockwise direction and the Snell's-law path R_R moves away from R to the right (see Figure 58).

Case 2: (See Figure 59.) R intersects B at point P so that its angle of incidence is measured in the clockwise direction from the normal to B. Suppose

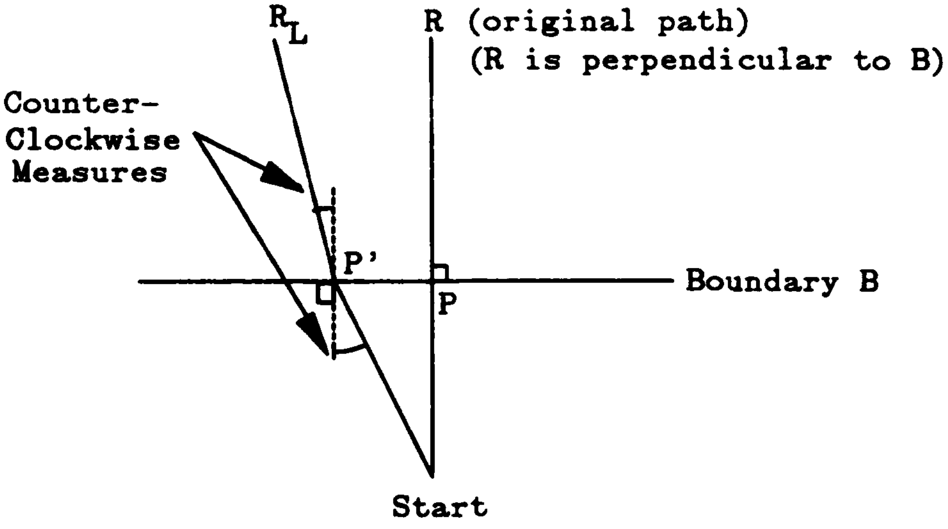


Figure 57. Illustration for Lemma 1, Case 1, A Counter-Clockwise Rotation, P' Left of P

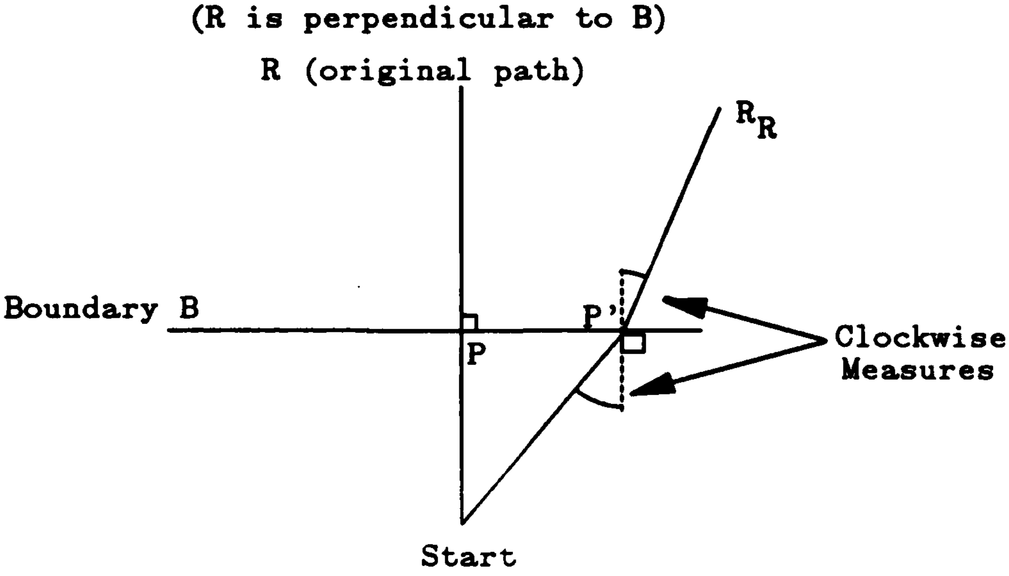


Figure 58. Illustration for Lemma 1, Case 1, A Colockwise Rotation, P' Right of P

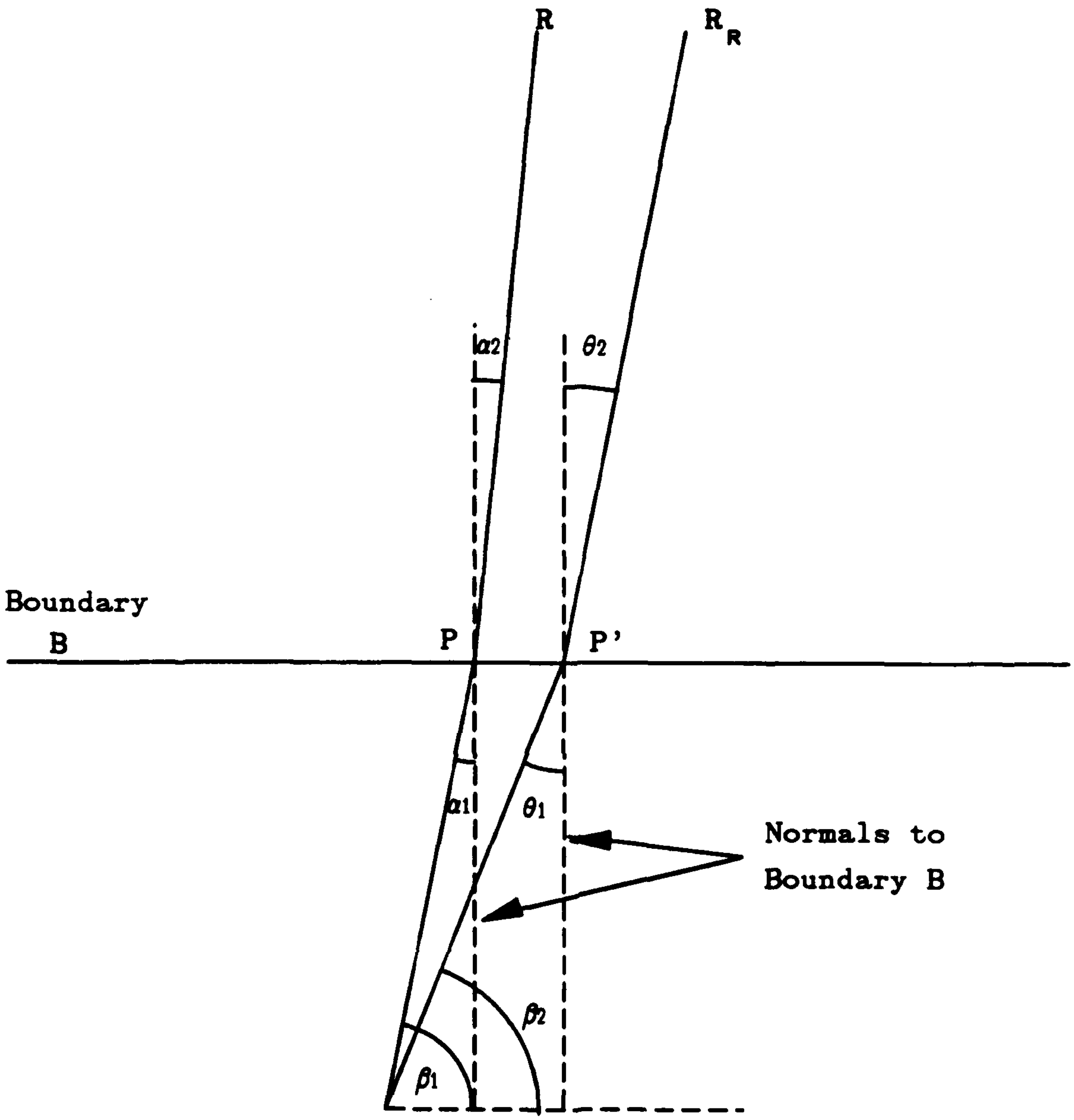


Figure 59. Example for Lemma 1, Case 2, Part 1 Where P' is to the Right of P and R has a Clockwise Measured Angle of Incidence With Boundary B

that some Snell's-law path R_R intersects B at point P' where P' is to the right of P. Clearly, if P' is to the right of P then

$$\beta_2<\beta_1$$

Note that:

$$\beta_1+\alpha_1=90,\ \beta_2+\theta_1=90$$

$$\beta_1+\alpha_1=\beta_2+\theta_1$$

$$\beta_2+\alpha_1<\beta_1+\alpha_1$$

$$\beta_2+\alpha_1<\beta_2+\theta_1$$

$$\alpha_1<\theta_1$$

According to Snell's law:

$$\frac{u_2\sin(\alpha_2)}{u_1}=\sin(\alpha_1)$$

$$\frac{u_2\sin(\theta_2)}{u_1}=\sin(\theta_1)$$

$$\alpha_1<\theta_1 \ \textit{implies} \ \sin(\alpha_1)<\sin(\theta_1)$$

$$\text{thus, } \frac{u_2\sin(\alpha_2)}{u_1}<\frac{u_2\sin(\theta_2)}{u_1},$$

$$\sin(\alpha_2)<\sin(\theta_2),$$

$$\alpha_2<\theta_2,$$

and therefore. R is left of R_R

Now suppose that some Snell's-law path R_L intersects B at P' where P' is to the left of P (see Figure 60). By the case above, Snell's-law path R lies entirely to the right of Snell's-law path R_L. Therefore, Snell's-law path R_L lies entirely to the left of Snell's-law path R.

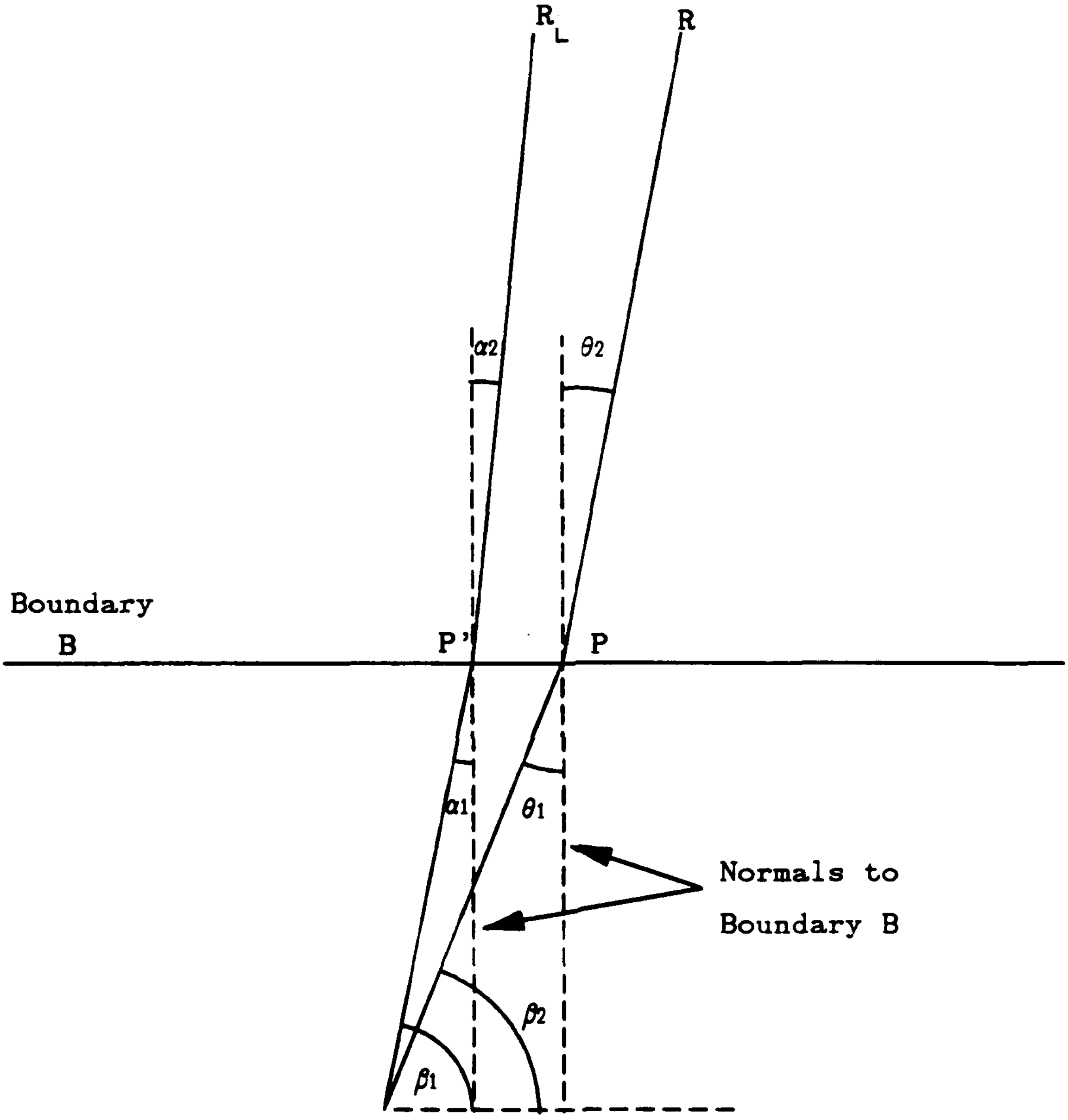


Figure 60. Example for Lemma 1, Case 2, Part 2 Where P' is to the Left of P and R has a Clockwise Measured Angle of Incidence with Boundary B

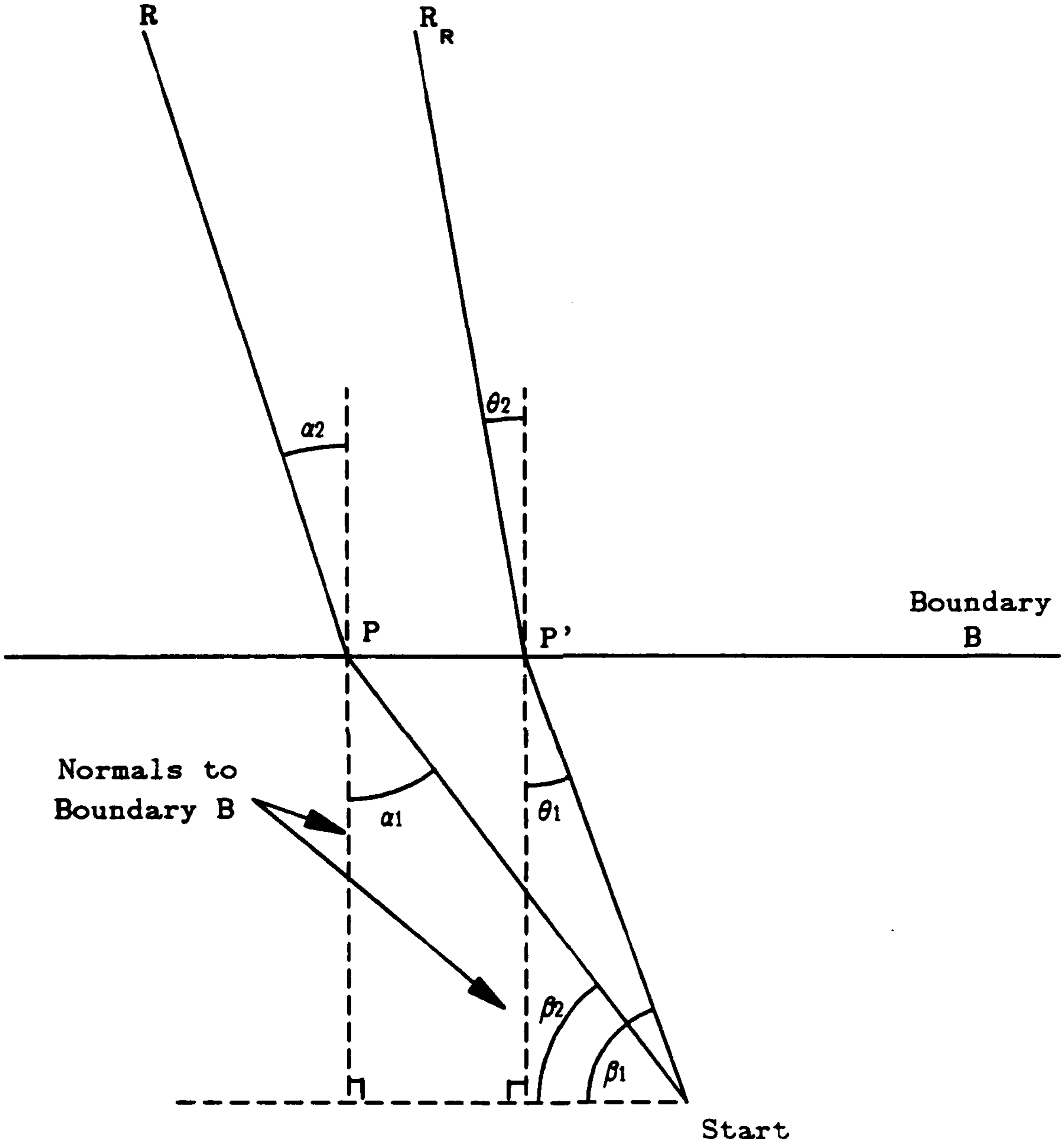


Figure 61. Example for Lemma 1, Case 3, Part 1 Where P' is to the Right of P and R has a Counter-Clockwise Measured Angle of Incidence with Boundary B

Case 3: (See Figure 61) Let Snell's-law path R intersect B at point P so that its angle of incidence with the normal to B is measured in a counter-clockwise direction. Suppose some Snell's-law path R_R intersects B at P' where P' is to the right of P, as in Figure 61. Clearly, if P' is to the right of P then $\beta_1 > \beta_2$. Thus,

$$\beta_2 + \alpha_1 = 90,\ \beta_1 + \theta_1 = 90$$

$$\beta_2 + \alpha_1 = \beta_1 + \theta_1$$

$$\beta_1 + \alpha_1 > \beta_2 + \alpha_1$$

$$\beta_1 + \alpha_1 > \beta_1 + \theta_1$$

$$\text{Thus, } \alpha_1 > \theta_1$$

Similar to the reasoning in case 2 above, $\alpha_2 > \theta_2$. Therefore, R_R is entirely to the right of R.

Suppose that some Snell's-law path R_L intersects B at P' where P' is to the left of P (see Figure 62). Then, by the case above, Snell's-law path R is entirely to the right of Snell's-law path R_L. Therefore, Snell's-law path R_L is entirely to the left of Snell's-law path R.

Therefore, by the three cases above, given any Snell's-law path R, intersecting boundary B at P, then any Snell's-law path R_R that intersects B at P' to the right of P, lies entirely to the right of R and any Snell's-law path R_L that intersects B at P' to the left of P lies entirely to the left of R. *QED*.

Corollary 1 to Lemma 1: Any two Snell's-law paths within the same 1-explored wedge do not intersect each other prior to their second boundary-crossing episodes.

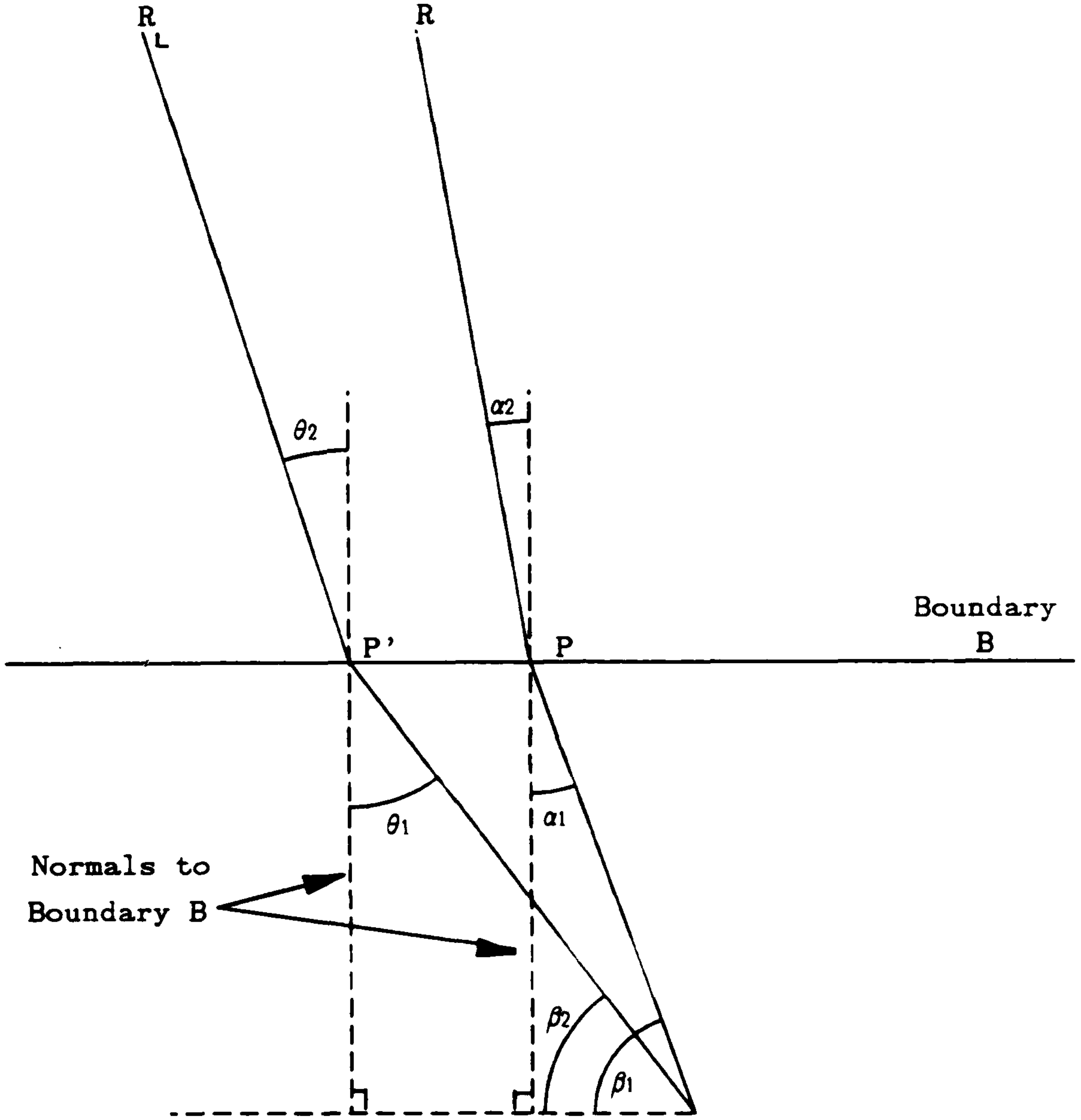


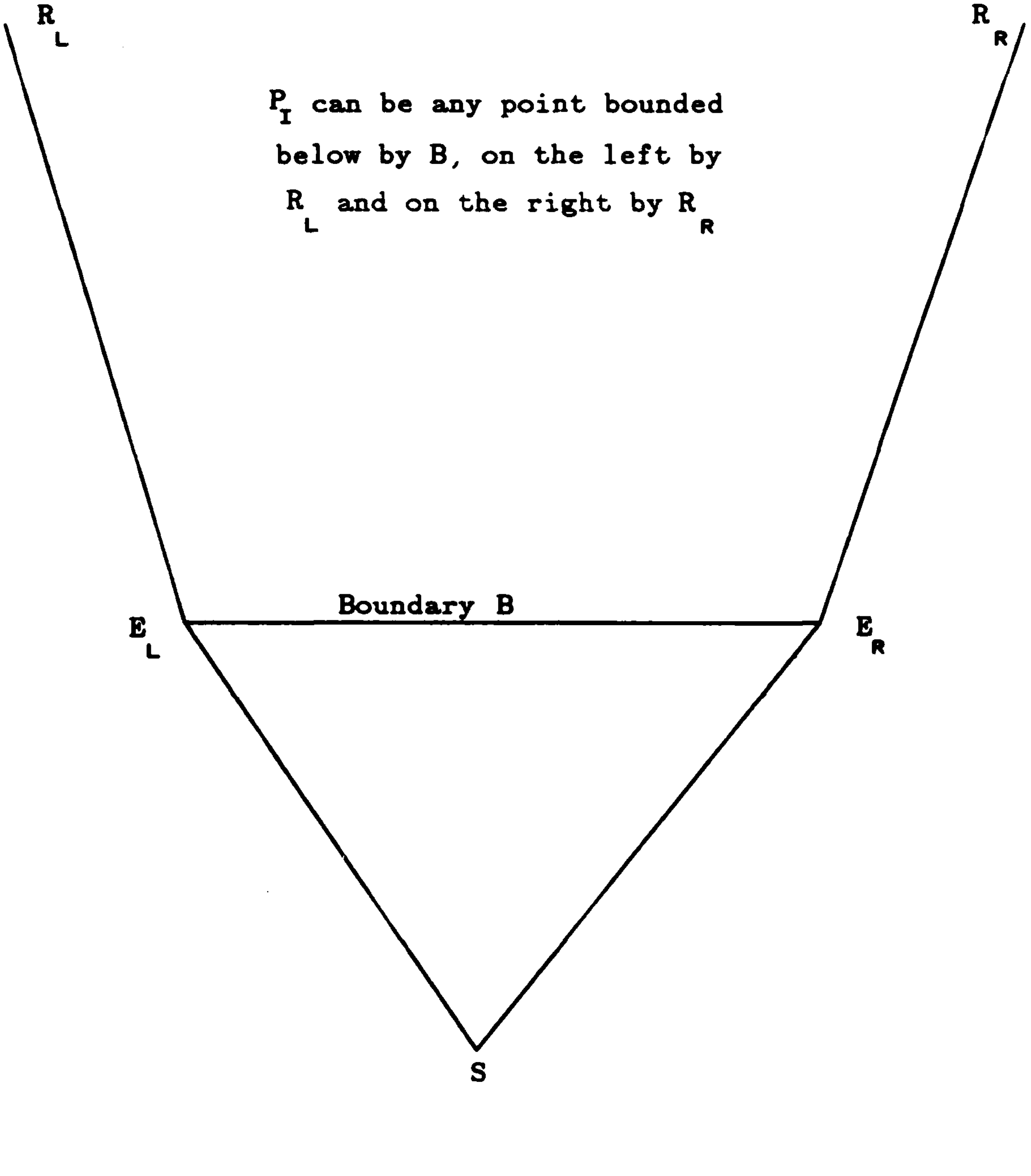
Figure 62. Example for Lemma 1, Case 3, Part 2 Where P' is to the Left of P and R has a Counter-Clockwise Measured Angle of Incidence with Boundary B

Proof: Two Snell's-law paths within the same 1-explored wedge must intersect the same homogeneous-cost region boundary at their first boundary-crossing episode. By Lemma 1, these Snell's-law paths cannot intersect prior to their second boundary-crossing episode. *QED.*

Corollary 2 to Lemma 1: Any two known-optimal-cost Snell's-law paths do not intersect each other more than once unless the cost of the two subpaths between the first and last intersection of the Snell's-law paths have equal subpath cost.

Proof: Assume Snell's-law path SL_1 is the single optimal-cost path between points A and B and that Snell's-law path SL_2 is the single optimal-cost path between points C and D. Assume SL_1 and SL_2 intersect at points P_1 and P_2. If the P_1-to-P_2 subpath along SL_1 has lower cost than the P_1-to-P_2 subpath of SL_2 then there is a C-to-D path that follows along SL_2 up to point P_1, then follows the P_1-to-P_2 subpath of SL_1, and then follows along path SL_2 from P_2 to D and this path has lower cost than the SL_2 C-to-D path. This is a contradiction since SL_2 is known to be the optimal-cost C-to-D path. Therefore, the two P_1-to-P_2 subpaths must have equal cost between points P_1 and P_2. *QED.*

Lemma 2: (Refer to Figure 63.) There is a Snell's-law path from point S to point P_I that intersects boundary B between the endpoints of B, (which are also region vertices) E_L and E_R, if and only if boundary B lies between points S and P_I and P_I is within the 1-explored portion of th: wedge (i.e., is interior to the polygon describing the wedge) formed by R_L and R_R where R_L is a Snell's-law path through the left endpoint of B (E_L) and R_R is a Snell's-law path through the right endpoint of B (E_R).



R_L
R_R
P_I can be any point bounded below by B, on the left by R_L and on the right by R_R
Boundary B
E_L
E_R
S

Figure 63. Example for Lemma 2, P_I is Within the Wedge Defined by R_L and R_R and Boundary B Lies Between P_I and S

Proof: (part 1 for Lemma 2) P_I is within the wedge, therefore, a Snell's-law path from S, across B to P_I exists.

By Corollary 1 to Lemma 1, the optimal-cost S-to-P_I path cannot intersect either R_L or R_R. Since P_I lies between these two wedge-defining Snell's law paths, the optimal-cost S-to-P_I path must intersect boundary B. As shown in II.C, the optimal-cost S-to-P_I path that intersects boundary B must obey Snell's law at the boundary-crossing episode. Therefore, a Snell's-law path from S to P_I exists.

(part 2 for Lemma 2) P_I is not within the wedge formed by R_L and R_R. Therefore, no Snell's-law path from S that intersects boundary B between E_L and E_R to P_I exists.

By Lemma 1, any Snell's-law path from S, intersecting B between E_L and E_R will be bounded on the left by R_L and on the right by R_R. Since P_I is either to the right of R_R or to the left of R_L, none of these Snell's law paths pass through the point P_I. Therefore, there is no path from S, across B between E_L and E_R to P_I such that this path obeys Snell's law at the boundary-crossing episode about boundary B. *QED.*

Lemma 3: (Refer to Figure 64.) If the goal P_G lies outside the wedge defined by Snell's-law paths R_L and R_R, then a minimum-cost path involving boundary B from S to P_G is S-to-E_j-to-P_G where E_j is the closest endpoint of B to P_G.

Proof: Movement of the path below B beyond either endpoint of B towards P_G is prohibited by definition since the path must intersect boundary B. Assume P_B is some point on B properly between E_L and E_R. Assume, as in Figure 64, that P_G is closest to E_R. Then, any path S-to-P_B-to-P_G must have greater cost than the

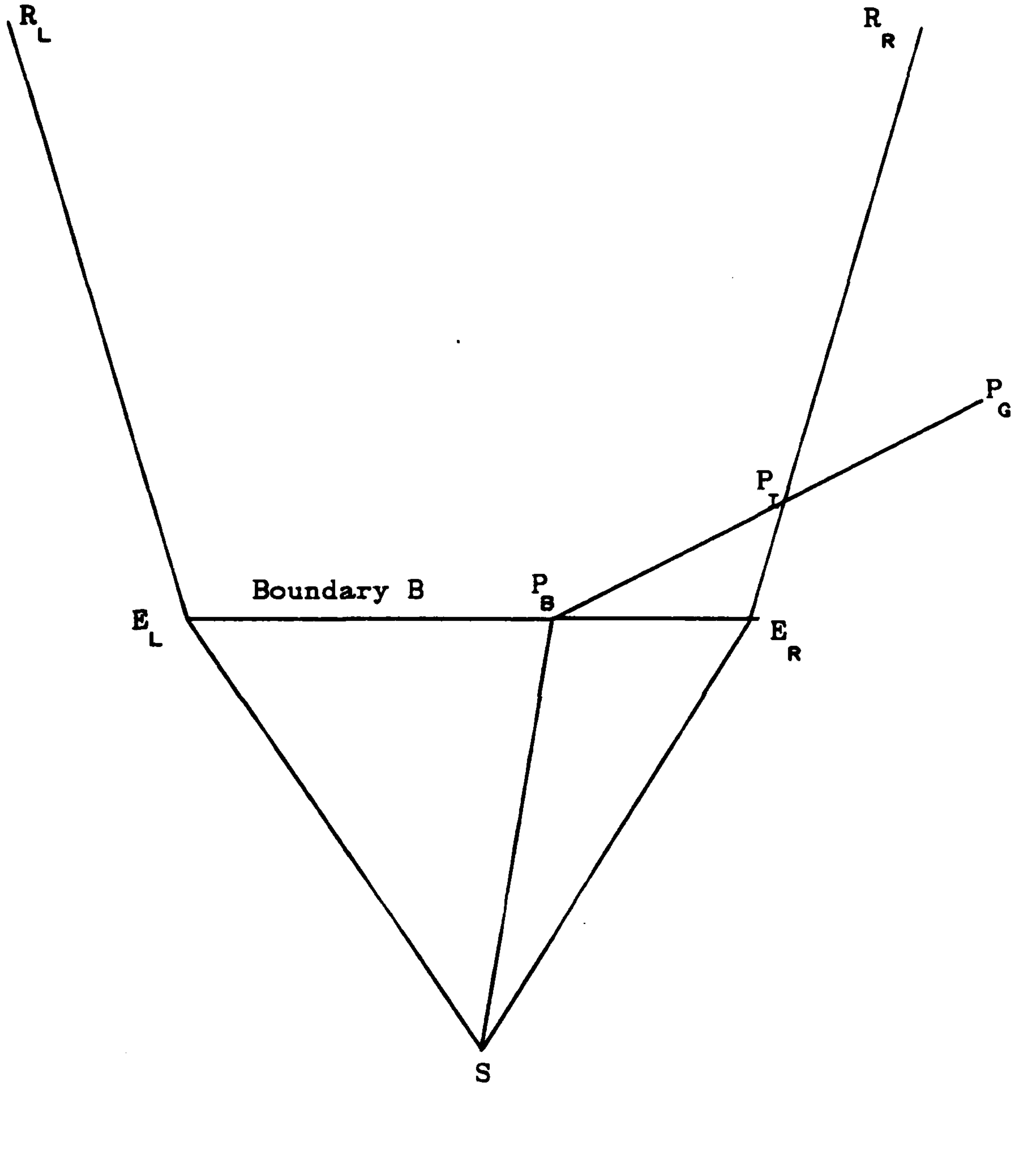


Figure 64. Example for Lemma 3, the Goal, P_G Lies Outside the Wedge Defined by R_L and R_R

path S-to-E_R-to-P_G. The former path must intersect the path R_R at some point, P_I. Since P_I is on R_R, the path S-to-P_B-to-P_I must have greater cost than the path S-to-E_R-to-P_I since the latter is a Snell's-law path between S and P_I and must therefore be a minimal-cost path between those two points. Therefore, the path S-to-P_B-to-P_I-to-P_G has greater cost than the path S-to-E_R-to-P_I-to-P_G. Also, the path E_R-to-P_G has lower cost than the path E_R-to-P_I-to-P_G since the former is a straight line (and thus has least distance) and the two paths have the same cost rate. Therefore,

$$cost(S-to-E_R-to-P_G) < cost(S-to-E_R-to-P_I-to-P_G)$$

$$cost(S-to-E_R-to-P_I-to-P_G) < (costS-to-P_B-to-P_I-to-P_G)$$

Clearly, the same proof technique applies to goal points closest to E_L. *QED.*

Extending the results of the preceding lemmas to an indefinite number of boundary-crossing episodes is accomplished by the following theorems. Theorem 1 extends the result of Lemma 1.

Theorem 1: Any two Snell's-law paths within a K-explored wedge defined by Snell's-law paths R_L and R_R do not intersect within the K-explored portion of the wedge.

Proof: (See Figure 65.) By Corollary 1 to Lemma, no two Snell's-law paths can intersect each either before or immediately after their first boundary-crossing episode. Thus, the theorem holds when K = 1. Assume that the theorem holds for the first N boundary-crossing episodes (where N = K - 1). Let the angle of refraction for R_L after intersecting the Nth boundary be θ_L. Let the angle of refraction for R_R after intersecting the Nth boundary be θ_R. (Both angles are in

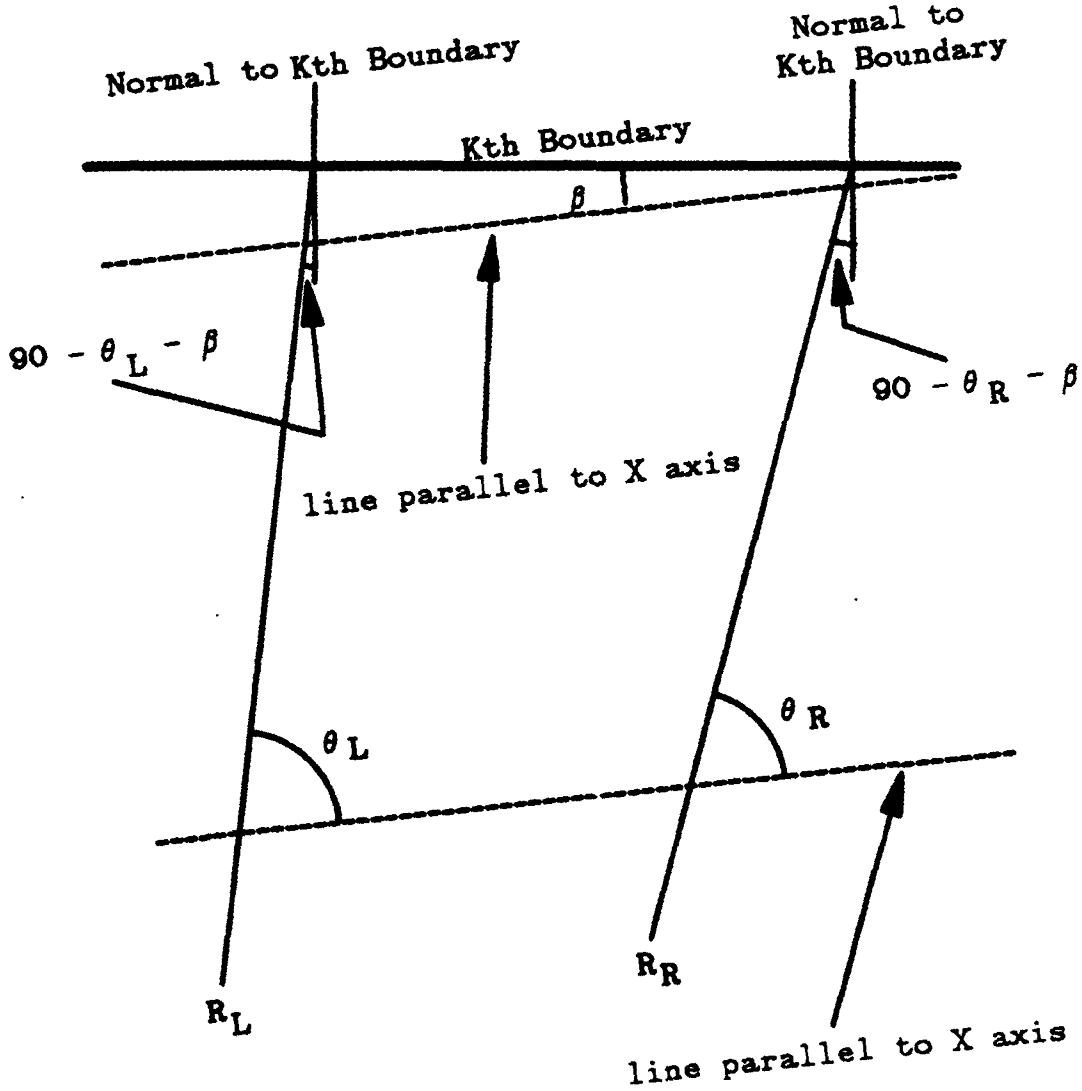


Figure 65. Illustration for Theorem 1
Snell's-law Paths Within The Same K-explored Wedge
Do Not Intersect

terms of the coordinate system X axis). Let the angle between boundary K and the X axis be β. It must be true that R_L has an angle of incidence with boundary K of $90 - \theta_L - \beta$ (in degrees). Similarly, R_R has an angle of incidence with boundary K of $90 - \theta_R - \beta$. Both Snell's-law paths R_L and R_R must obey Snell's law at their Kth boundary-crossing episode. Thus, the sine of their angles of refraction after intersecting boundary K must equal the sine of their angles of incidence multiplied by a common fraction (the ratio of cost rates about boundary K). It should be clear that when θ_L and θ_R do not allow R_L and R_R to intersect (before intersecting boundary K), then the angles of refraction computed after intersecting boundary K (as above) will also be such that R_L and R_R cannot intersect each other after their Kth boundary-crossing episode. Thus, the theorem holds for the Kth boundary-crossing episode where K = N + 1 and Theorem 1 is established. *QED.*

Theorem 2 extends the result of Lemma 2.

Theorem 2: There is a Snell's-law path within a K-explored wedge to the closest unsolved search point within that wedge.

Proof: By Lemma 2, this is true when K = 1. Assume that the theorem holds for the first N boundary-crossing episodes within the wedge where N = K - 1. Thus, there is a path from the start to every point on the Kth boundary (within the wedge) since these points on boundary K are all within the N-explored portion of the wedge. By Theorem 1, we know that no two Snell's-law paths within the K-explored portion of the wedge intersect. By definition, the closest unsolved search point lies beyond boundary K and between the left and right wedge-defining

Snell's-law paths, R_L and R_R. Thus, tracing a new Snell's-law path through the K-explored portion of the wedge must result in a Snell's-law path that lies between R_L and R_R and the unsolved search point must be between either the new path and R_L or between the new path and R_R. Since every point on boundary K is on a Snell's-law path, the tracing of new paths through the wedge can be continued until the search point is bracketed between two Snell's-law paths that pass through infinitesimally separated points on boundary K. Eventually, one of the new Snell's-law paths must pass through the closest unsolved search point (that is, by definition, located beyond boundary K). Thus, the theorem holds for the K = N + 1 case. This establishes Theorem 2. *QED.*

The following Theorem extends the results of Lemma 3 to encompass K-explored wedges.

Theorem 3: If the goal lies outside a wedge, then there is no Snell's-law path from the wedge tip to the goal such that the path lies entirely within the wedge.

Proof: Let Snell's-law paths R_L and R_R define the left and right wedge boundaries. Since the goal lies outside the wedge, then any Snell's-law path within the wedge must intersect either R_L or R_R in order to pass through the goal. By Theorem 1, this cannot occur. *QED.*

The following two lemmas apply directly to Snell's-law paths that include arbitrary numbers of boundary-crossing episodes.

Lemma 4: Given Snell's-law paths R_L and R_R that define a K-explored wedge, then the minimum-cost path from the wedge tip to the Kth boundary is either from the tip along R_L to boundary K, from the tip along R_R to boundary K, or along a new Snell's-law path, R_N that starts at the wedge tip and intersects boundary K at the normal. (Also see [Ref. 3, p.25] for proof of a similar property.)

Proof: There are two cases; either R_L and R_R have the same direction of rotation (i.e., either clockwise or counter-clockwise) as they are refracted by intersection with boundary K or they have different directions of rotation after intersecting boundary K.

Case 1: R_L and R_R turn (or rotate) in the same direction after intersection with boundary K (refer to Figure 66). Consider the point A on path R_A. By definition of a Snell's-law path, R_A must be the minimum-cost path from the wedge tip, S, to A. However, note that the shortest-distance path from boundary K to point A is a normal to K that intersects K at the same point as Snell's-law path R_L. The normal to K through A is in the same homogeneous-cost region as the portion of R_A above K and therefore has lower path cost from K to A. Since R_A is the minimum-cost path from S to A it must be true that R_A has a lower cost to reach K than does path R_L (up to boundary K). A similar argument can be made for any point on the dashed A to C line segment of Figure 66, resulting in the fact that R_R must be the locally minimum-cost path through the K-explored portion of the wedge (up to boundary K). The same general proof technique holds when all paths exiting boundary K rotate in a clockwise direction. In this case, path R_L is the minimum-cost path through the K-explored portion of the wedge.

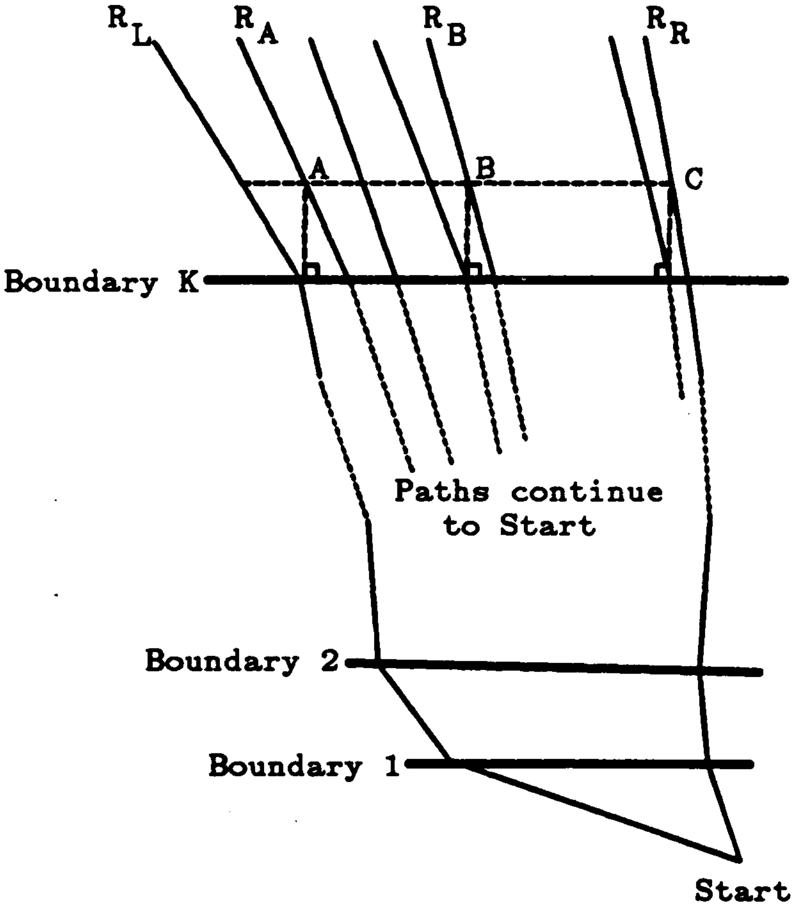


Figure 66. Example for Lemma 4, Case 1

Paths Have the Same Rotation Direction

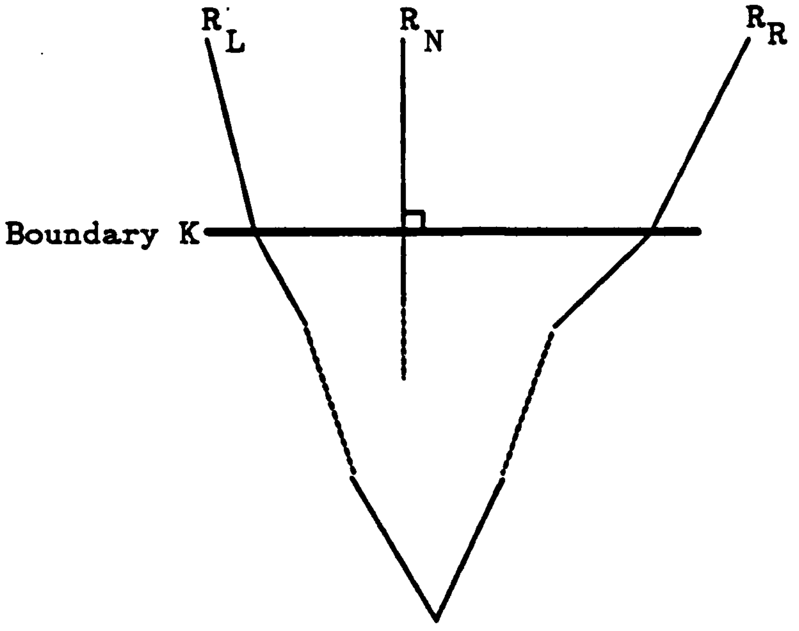


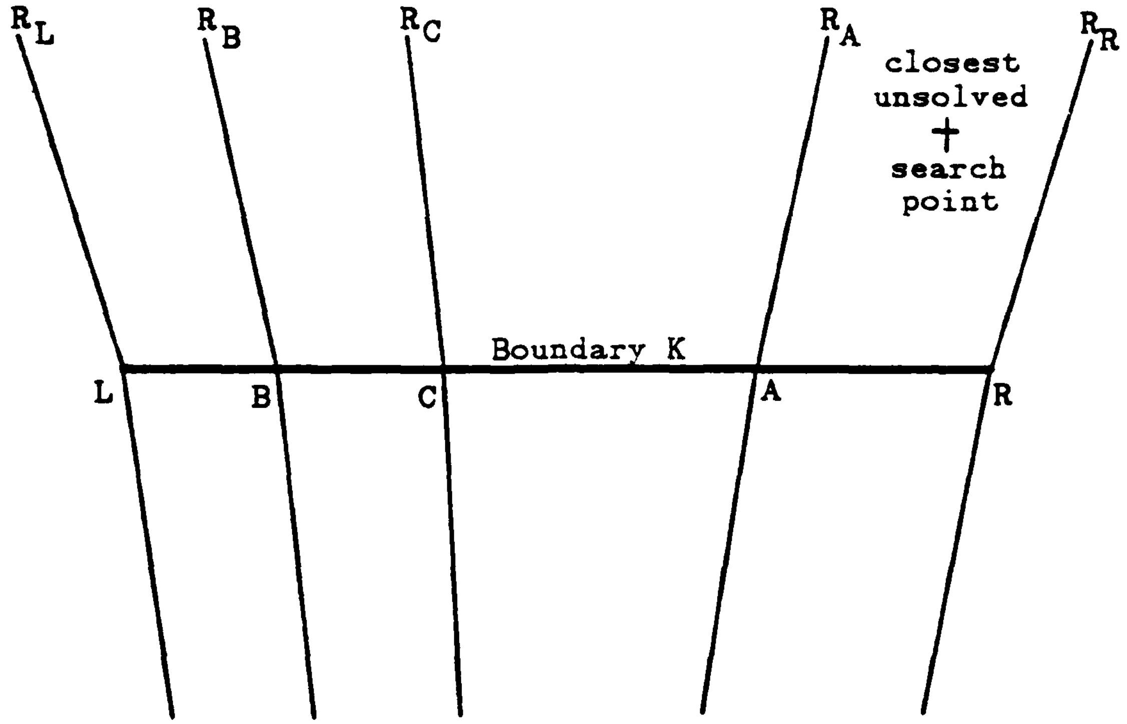
Figure 67. Example for Lemma 4, Case 2
Paths Have Opposite Rotation Directions

Case 2: R_L and R_R rotate in different directions after intersecting boundary K (refer to Figure 67). By convexity, there must be some path, R_N, between R_L and R_R, that intersects K at normal. By case 1, path cost must be monotonically decreasing from R_L to R_N. Similarly, path cost must be monotonically decreasing from R_R to R_N. Therefore, R_N must constitute the minimum-cost path through the K-explored portion of the wedge. *QED.*

Lemma 4 is useful in forming lower-bound evaluations for the cost of any path through a K-explored wedge. Lemma 5, below, establishes a criterion that is useful in producing an iterative search strategy to solve Snell's law, given two points that we wish to connect by a Snell's-law path. Before introducing Lemma 5, we note that each Snell's-law path within a K-explored portion of a wedge is uniquely determined by any point on that path (by Theorem 1). Thus, any Snell's-law path within a K-explored portion of a wedge is uniquely determined by a point on the Kth boundary within the wedge. Let P be the distance from such a point to the point of the Kth boundary-crossing episode of the Snell's-law path that defines the wedge left boundary. Since there is a unique minimum distance from a point to a line segment, there is a unique minimum distance from the closest unsolved search point within a wedge to the path segment (immediately after crossing the Kth boundary) of a Snell's-law path. We define a function $dist(P) = D_P$ as a mapping from a Snell's-law path-defining point characterized by P (as above, a distance along the Kth boundary) to the minimum distance from the closest unsolved search point to that path segment of the Snell's-law path immediately after the Kth boundary-crossing episode.

Lemma 5: $dist(P) = D_P$ is a quasi-convex function of P.

Proof: (Refer to Figure 68.) Let $dist(A) = D_A$ and $dist(B) = D_B$. The definition of quasi-convexity requires that $dist(\lambda A + (1-\lambda)B) \leqslant \max\{D_A, D_B\}$ for each $\lambda \epsilon [0,1]$. Let $A > B$, $D_A \geqslant D_B$ and let there be some point C distance away (along the Kth boundary) from the Kth boundary-crossing episode of the Snell's-law path that defines the wedge left boundary such that C lies between A and B (i.e., $C = \lambda A + (1-\lambda)B$) and $dist(C) = D_C$. Let R_A, R_B, and R_C be the Snell's-law paths determined by A. B. and C respectively. If the closest unsolved search point within the wedge lies between R_A and R_B. then $D_C \leqslant D_A$ since Snell's-law paths within a K-explored wedge cannot intersect each other (by Theorem 1). Suppose then that the closest unsolved search point does not lie between R_A and



These paths continue through the wedge back to the start

Figure 68. Dist(P) Quasi-Convexity

R_B and assume that $D_C > D_A$. If $D_C > D_A$ then the Snell's-law path determined by A (i.e., R_A) must lie between the closest unsolved search point and the Snell's-law path determined by C (i.e., R_C). For this to be the case, the problem geometry must be similar to that depicted in Figure 68 (since, again by Theorem 1, none of the Snell's-law paths can intersect each other within the K-explored portion of the wedge). In this case, $D_B < D_A$, which is a contradiction. Therefore, it must be true that $D_C < D_A$ and *dist*(P) is a quasi-convex function. *QED.*

F. RELATIONS BETWEEN ADJACENT WEDGES

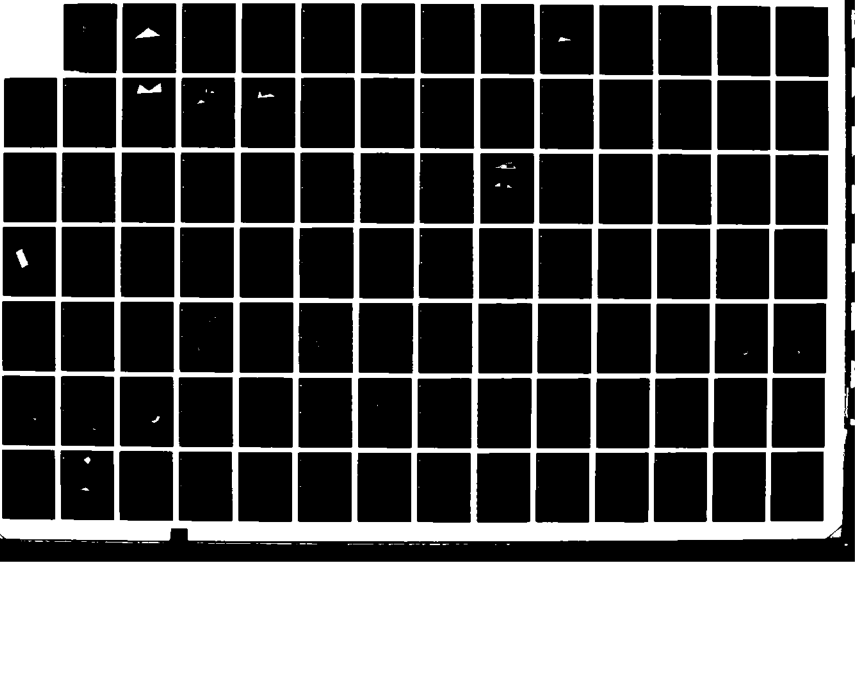
The preceding discussions all focus on Snell's-law path characteristics within single wedges. One way to view wedges is as areal divisions of physical space (as represented by the area-cost map). Thus, they are not isolated. Each wedge has an adjacent wedge as a neighbor. Suppose that wedge W is defined with Snell's-law path R_L as its left boundary and R_R as its right boundary. Let P be the closest unsolved search point within W. Once a Snell's-law path within W to P is found, P becomes a solved search point. Further, the Snell's-law path to P can be used to define new wedges (see Figure 69). Since P is a vertex of a homogeneous-cost region, there are two line segments (modeling region boundaries) that share P as an endpoint (for example, sides B_1 and B_2 in Figure 69 have P as a common endpoint). The Snell's-law path to P can be continued through P in two ways. In Figure 69, R_{PL} is a new Snell's-law path that intersects side B_1 of the high-cost regior Path R_{PR} is a Snell's-law path through P on side B_2 of the region. Three new wedges can be formed from these two new paths, all of which *refine* (make new sub-wedges from) the original wedge defined by R_L and R_R. One new wedge, W1, has R_L as a left boundary and R_{PL} as a right boundary. Similarly, wedge W3 is defined by R_{PR} on the left and R_R on the right. The third wedge, W2, has the Snell's-law path from S to P as an approach path and is defined thereafter as having R_{PL} on its left and R_{PR} on its right. This wedge is empty since it terminates immediately when R_{PL} and R_{PR} intersect at vertex P.

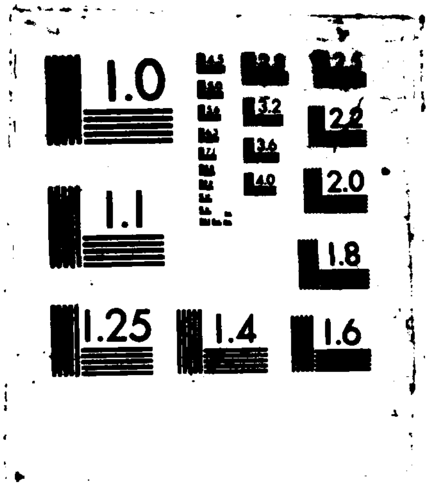
Before justifying the emptiness of W2, consider wedges W1 and W3 and their relation to the points P1 through P5 in Figure 69. From Theorem 2 and Theorem

AD-A183 900 SOLVING A CLASS OF SPATIAL REASONING PROBLEMS: MINIMAL-COST PATH PLANNING IN THE CARTESIAN PLANE(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA R F RICHBOURG JUN 87

3/5

UNCLASSIFIED F/G 12/9 NL




1.0
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

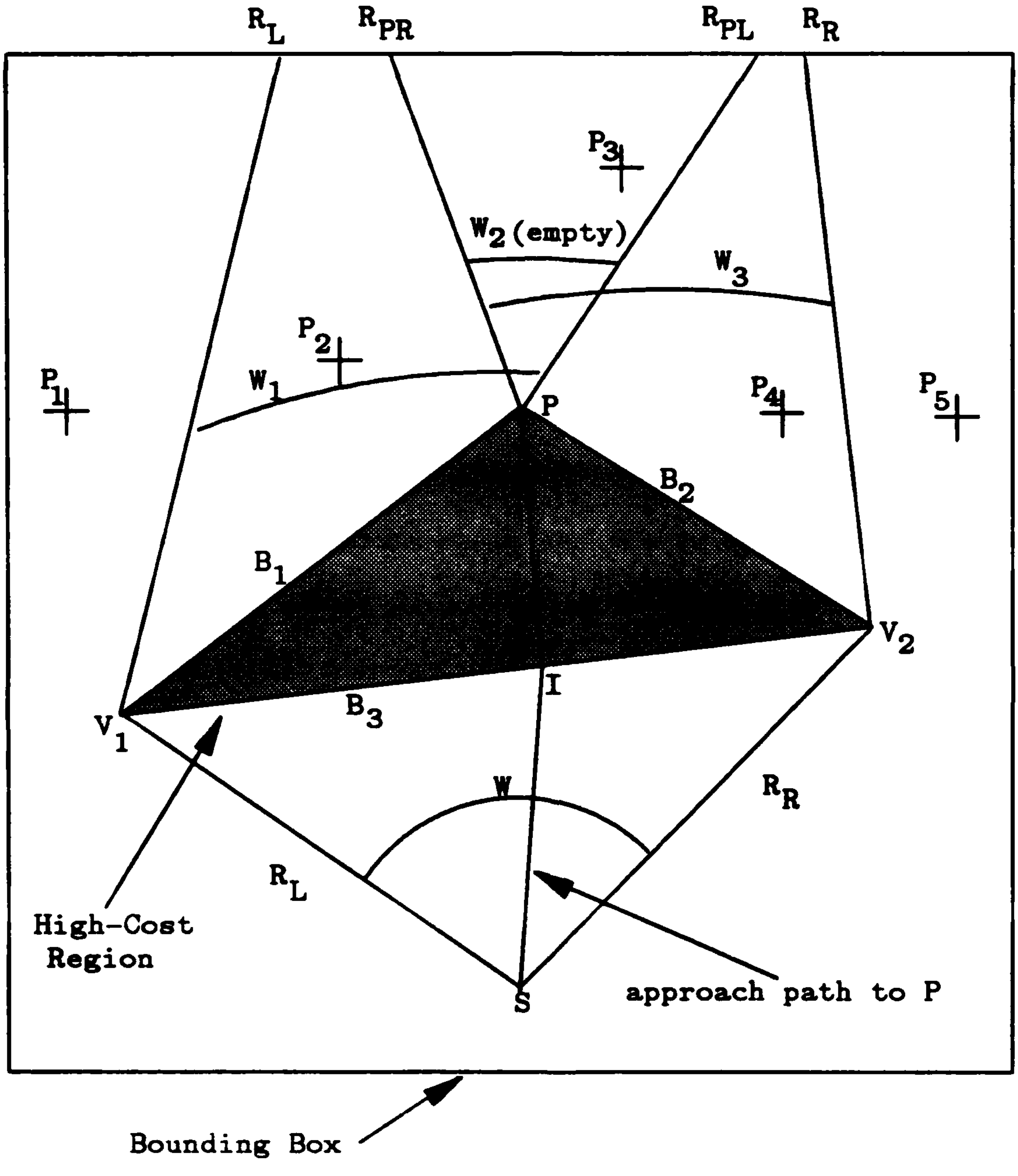


Figure 69. Example of Refining a Wedge Into Sub-wedges

3 it is clear that the least-cost path from S to P1 goes through vertex V1 of the high-cost region (but the path does not intersect the high-cost region). Similarly, the least-cost S-to-P5 path is S-V2-P5. The point P2 is only included in the W1 wedge. Based on Theorems 2 and 3, the globally minimum-cost path from S to P2 is a Snell's-law path intersecting sides B_3 and B_1 of the high-cost region. Similarly with point P4, wedge W3, and sides B_3 and B_2 of the high-cost region.

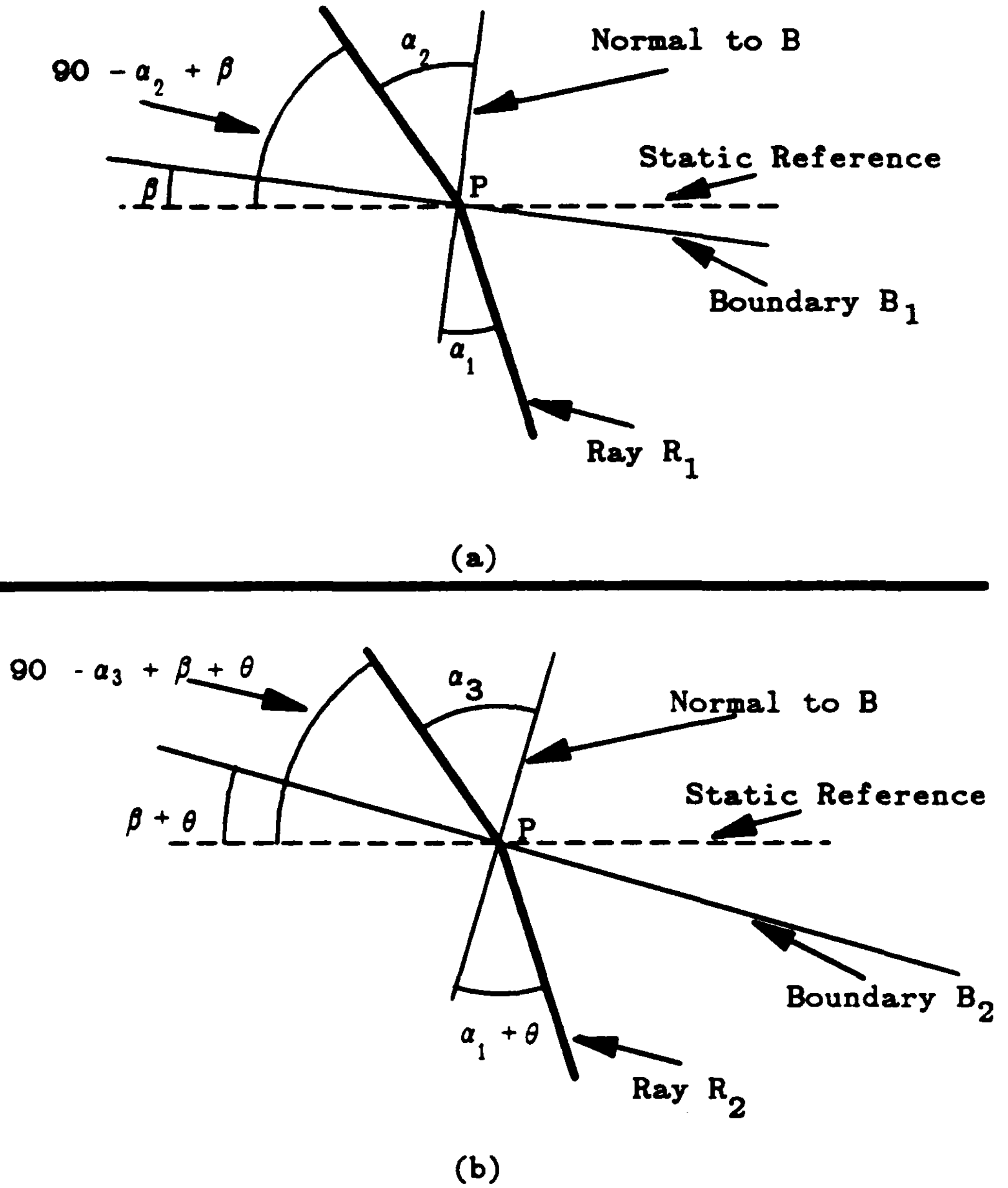
The point P3 is inside both the W1 and W3 wedges. Theorems 2 and 3 imply that the cost of the S-V1-P3 path is greater that the cost of a Snell's-law path from S to P3 across sides B_3 and B_1 of the high-cost region. Similarly, there is a Snell's-law path from S to P3 that is within the W3 wedge. Thus, there are two locally-minimum-cost paths from S to P3. In general, it is difficult to know, a priori, which of the two local minima is a global minimum. We must find both paths and compare their costs.

The W2 wedge must be empty, even though it appears to contain point P3 of Figure 69. Consider a path from S to P3 that includes S-I-P as an approach path. The least-cost such path is S-I-P-P3. By Theorem 2, there is a Snell's-law path through wedge W3 that intersects B_3 to the right of I and B_2 to the right of P and this path has lower cost than the S-I-P-P3 path. A similar statement can be made concerning a Snell's-law path through wedge W1. Thus, for any point between R_{PL} and R_{PR}, the optimal-cost path from S to that point is either in wedge W1 or in wedge W3.

Wedge W2 is empty because its left boundary, R_{PL}, lies to the right of its right boundary, R_{PR}. These two paths intersect at point P, immediately terminating the wedge. The relation between R_{PL} and R_{PR} exists because because the continuation of the S-I-P path exits an high-cost region through a region vertex forming an angle greater than 180 degrees. Lemma 6 below formally establishes the relation. Before considering Lemma 6, we introduce some terminology. A polygon in which no two adjacent polygon boundaries form a reflex angle (i.e., an angle between 180 and 360 degrees, also known as a concave angle) relative to the interior of the polygon is termed a *convex polygon.* By definition, each angle in a convex polygon is a *convex angle* formed about a *convex vertex* of the polygon. If one or more angles of the polygon does form a reflex angle, then the polygon is a *non-convex polygon* and the angles that are reflex angles are said to be *non-convex angles.* A non-convex angle is formed about a *non-convex vertex* of the polygon. Finally, a Snell's-law path can be *split* at a region vertex, P, resulting in two Snell's law paths that are identical from the wedge tip to P and different thereafter. The two Snell's-law path that result from the split can be used to refine a wedge. As an example, the two Snells'-law paths resulting from a split at vertex P form a middle sub-wedge where P is the wedge tip.

Lemma 6: The splitting of a Snell's-law path at a convex vertex, P, of a high-cost region results in the creation of an empty middle sub-wedge. This sub-wedge is empty because the Snell's-law paths that define the sub-wedge left and right sides intersect each other at vertex P.

Proof: (Refer to Figure 70.) Let the dashed lines in Figure 70 represent a static reference line (such as the coordinate system X axis). Boundary B is rotated clockwise about this line by β in Figure 70a and by $\beta+\theta$ in Figure 70b. Assume



Note: Boundary B_2 is the adjacent region boundary to B_1 and B_2 is in a clockwise direction from B_1

Figure 70. Example for Lemma 6, Splitting a Snell's-law Path at a Convex Vertex of a High-Cost Region Results in the Creation of an Empty Middle Sub-wedge

that the figures can be overlaid so that the reference lines (as well as the bottom portions of the rays) are collinear. Consider the inequality

$$90-\alpha_3+\beta+\theta < 90-\alpha_2+\beta$$

If this inequality is true, then clearly, path R_2 lies in a counter-clockwise direction from path R_1. If the two angles are equal, the paths are identical. If the inequality expresses a greater than relation, then R_2 lies in a clockwise direction from R_1.

$$90-\alpha_3+\beta+\theta ?< 90-\alpha_2+\beta$$

$$-\alpha_3+\theta ?< -\alpha_2$$

$$\alpha_3-\theta ?> \alpha_2$$

$$\alpha_3 ?> \alpha_2+\theta$$

Since these are angular measures, an equivalent question is

$$\sin(\alpha_3) ?> \sin(\alpha_2+\theta)$$

Let U_1 and U_2 be cost measures such that the portions of R_1 and R_2 above B are traversed at cost U_2 and the portions of R_1 and R_2 immediately below B have cost rate U_1. Then, using Snell's law

$$\frac{U_2\sin(\alpha_1)}{U_1}=\sin(\alpha_2)$$

$$\frac{U_2\sin(\alpha_1+\theta)}{U_1}=\sin(\alpha_3)$$

Then, substituting

$$\sin(\alpha_3) ?> \sin(\alpha_2-\theta)$$

$$\equiv\frac{U_2\sin(\alpha_1+\theta)}{U_1} ?> \sin(\alpha_2+\theta)$$

by identity

$$\equiv \frac{U_2}{U_1}(\sin(\alpha_1)\cos(\theta)+\cos(\alpha_1)\sin(\theta)) ?> \sin(\alpha_2)\cos(\theta)+\cos(\alpha_2)\sin(\theta)$$

$$\equiv \frac{U_2}{U_1}(\sin(\alpha_1)\cos(\theta)+\cos(\alpha_1)\sin(\theta)) ?> \frac{U_2}{U_1}(\sin(\alpha_1))\cos(\theta)+\cos(\alpha_2)\sin(\theta)$$

$$(U_2/U_1)\cos(\alpha_1)\sin(\theta) ?> \cos(\alpha_2)\sin(\theta)$$

$$(U_2/U_1)\cos(\alpha_1) ?> \cos(\alpha_2)$$

$$U_2/U_1\left(1-\sin^2(\alpha_1)\right)^{1/2} ?> \left(1-\sin^2(\alpha_2)\right)^{1/2}$$

$$(U_2/U_1)^2(1-\sin^2(\alpha_1)) ?> 1-\sin^2(\alpha_2)$$

$$(U_2/U_1)^2-(U_2/U_1)^2\sin^2(\alpha_1) ?> 1-(U_2/U_1)^2\sin^2(\alpha_1)$$

$$(U_2/U_1)^2 ?> 1$$

$$(U_2/U_1) ?> 1$$

Thus, the relation between the two paths depends on the ratio of the costs in the adjacent regions.

If a region can be modeled by a convex polygon and we examine consecutive polygon boundaries in clockwise order, then every boundary must be rotated clockwise from its immediate predecessor boundary. Using this scheme, the boundary in Figure 70b would be the clockwise successor of the boundary in Figure 70a. Thus, in forming adjacent wedges, the path in Figure 70a would correspond to R_R and the path in Figure 70b corresponds to path R_L. Thus, these paths must intersect immediately when exiting through a convex vertex of an high-cost region. If exiting through a convex vertex of an low-cost region, then the inequality

$$90-\alpha_3-\beta+\theta > 90-\alpha_2+\beta$$

holds and the adjacent wedge-defining Snell's-law paths cannot intersect. Note that if the region vertex is non-convex, then the path in Figure 70a corresponds to a left wedge boundary (i.e., R_L) and the path in Figure 70b corresponds to a right wedge boundary. Thus, non-convexity of a region vertex may also reverse the intersection relation for adjacent wedge-defining Snell's-law paths. That is, exiting through a non-convex vertex of an low-cost region allows the same relations as exiting through a convex vertex of an high-cost region. *QED*.

Lemma 6 establishes three other related lemmas.

Lemma 7: The splitting of a Snell's-law path at a non-convex vertex, P, of a high-cost region results in the creation of a (possibly) non-empty middle sub-wedge since the two wedge-defining Snell's-law paths do not intersect each other at vertex P.

Lemma 8: The splitting of a Snell's-law path at a non-convex vertex, P, of a low-cost region results in the creation of an empty middle sub-wedge. This sub-wedge is empty because the Snell's-law paths that define the sub-wedge left and right sides intersect each other at vertex P.

Lemma 9: The splitting of a Snell's-law path at a convex vertex, P, of a low-cost region results in the creation of a (possibly) non-empty middle sub-wedge since the two wedge-defining Snell's-law paths do not intersect each other at vertex P.

G. PRUNING CRITERIA

Lemmas 6 and 8 define a pruning criterion: Wedges whose wedge tips correspond to convex high-cost region vertices or non-convex low-cost region vertices are always empty, and thus never need to be created or examined. Therefore, wedges that conform to the criteria of Lemmas 6 or 8 can always be eliminated *(pruned)* from the search space. There are other occasions when wedges can be pruned. The first of these depends on the establishment of Lemma 10.

Lemma 10: A globally optimal-cost solution path, P, between a given start, S, and goal, G, includes (as a portion of P) globally optimal-cost paths between any two points on P.

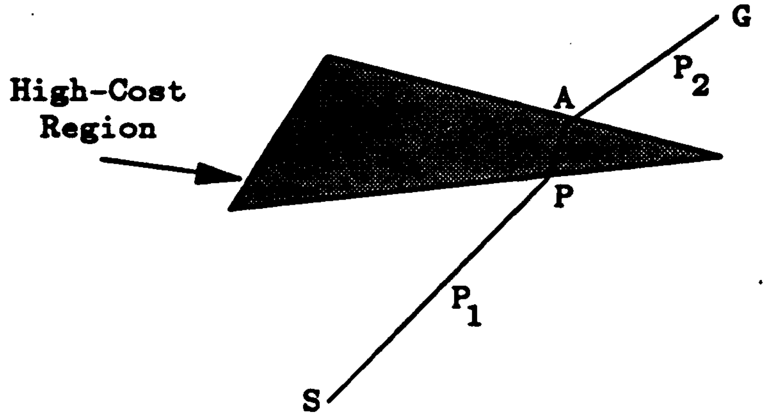


Figure 71. An S-to-G Solution Path

Proof: (Refer to Figure 71.) Suppose that the (S-P-A-G) path is known to be the globally minimum-cost path between S and G. Let P_1 and P_2 be any two points on path S-P-A-G between S and G. Denote the S-P-A-G path as S-P_1-P_2-G. Assume there is some path between P_1 and P_2, P_{min}, such that P_{min} has a lower cost than the cost from P_1 to P_2 along the S-P_1-P_2-G path. Then path S-P_{min}-G must have lower path cost than the S-P-A-G path. However, the path S-P-A-G is known to be the optimal-cost path from S to G. Therefore, by contradiction, P_{min} cannot exist. *QED.*

Now consider two distinct wedges, W_1 and W_2, that both have wedge tips at the same vertex, V. If V is not the start, then W_1 and W_2 must both have S-to-V approach paths and these approach paths must be different (otherwise W_1 is identical to W_2). Based on Lemma 10, the wedge having the higher-cost approach path from S to V can be pruned from the search space. This holds, regardless of what happens to W_1 and W_2 after V, even when the wedge having a higher-cost approach path contains a feasible start-to-goal solution and the other wedge does not. Any path through the wedge having the higher-cost approach path can be "shortcut" from S to V and thus cannot contain the optimal-cost solution path to a weighted-region problem.

In some cases, possible paths through specific pairs of sides of homogeneous-cost regions can also be pruned. That is, suppose that any Snell's-law path through a given wedge must intersect sides B_1 and B_2 of a high-cost region. Then, if sides B_1 and B_2 meet the criterion established by Lemma 11, the wedge can be pruned from the search space.

Lemma 11: (Refer to Figure 72.) Given cost rate U_2 inside a high-cost region polygon and cost rate U_1 outside the region, then no optimal-cost path intersects sides B_1 and B_2 of the region (consecutively) if all angles formed by the sides of the high-cost region between B_1 and B_2 are convex angles, sides B_1 and B_2

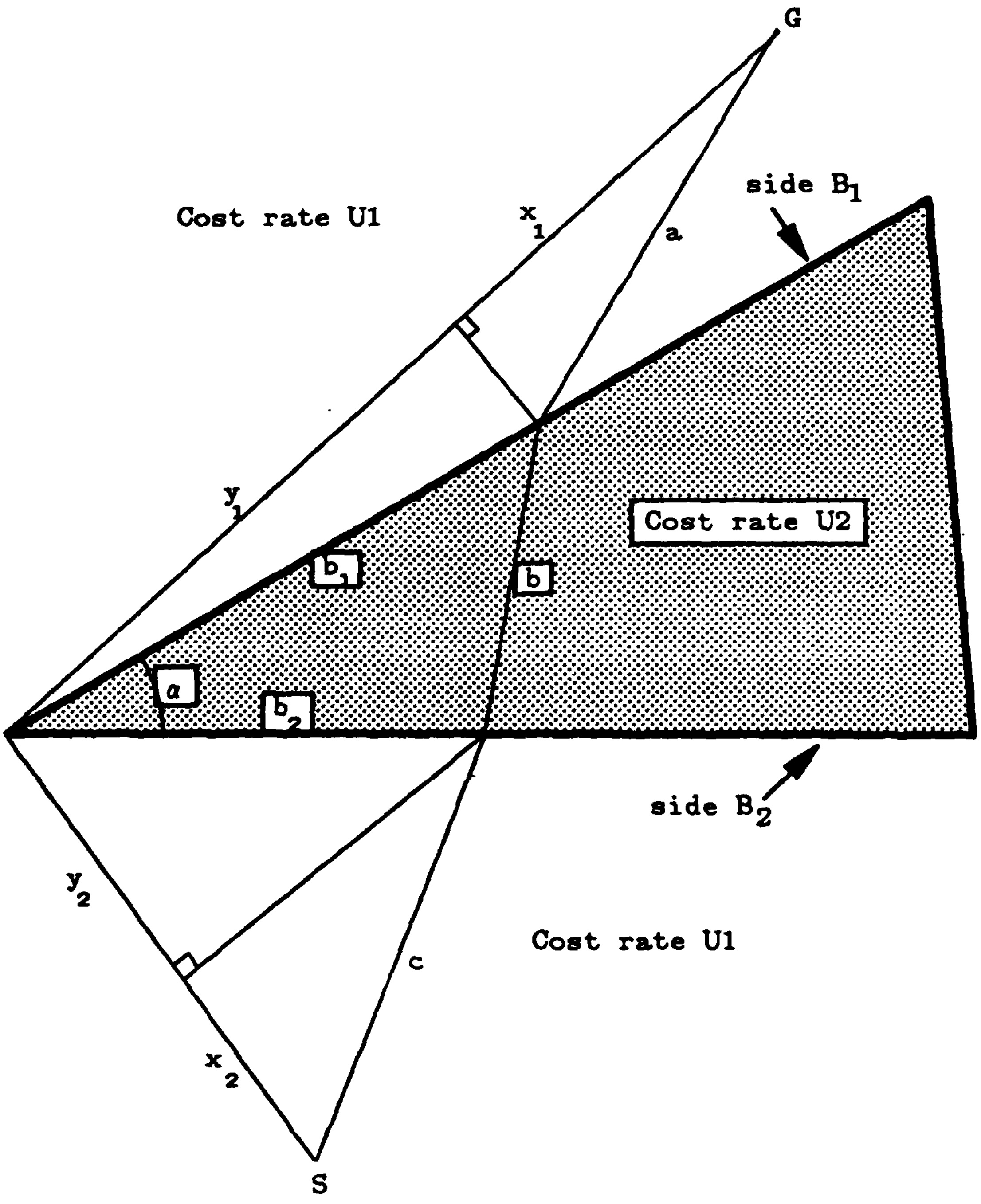


Figure 72. Example for Lemma 11

together form angle α and $\sin(0.5\alpha) > U_1/U_2$. ($B_1$ and B_2 form angle α when we consider B_1 and B_2 to be infinite length lines, not segments, that must intersect and α is measured at the point of intersection relative to the interior of the high-cost region.) In this case, there is some path that travels in the lower-cost region around sides B_1 and B_2 and has lower cost than any path that intersects the two sides.

Proof: We are comparing the costs of the line segments (x_1, y_1, y_2, x_2) and (a, b, c) as in Figure 72. By the Theorem of Pythagoras:

$$x_1 < a$$

$$x_2 < c$$

$$y_1 < b_1$$

$$y_2 < b_2$$

Thus, if $bu_2 > (b_1 + b_2)u_1$, the cost of going along (a, b, c) is greater that the cost associated with (x_1, y_1, y_2, x_2).

Transform the inequality $bu_2 > (b_1 + b_2)u_1$ into $\dfrac{b}{(b_1 + b_2)} > \dfrac{u_1}{u_2}$. Assuming the cost coefficients are known, then the ratio $\dfrac{u_1}{u_2}$ is known.

(Refer to Figure 73.) We first prove that $\dfrac{b}{(b_1 + b_2)}$ is minimized when $b_1 = b_2$.

First, use the law of sines to transform this problem into a more appropriate form.

$$\frac{b}{\sin(\alpha)} = \frac{b_1}{\sin(\beta_1)} = \frac{b_2}{\sin(\beta_2)}$$

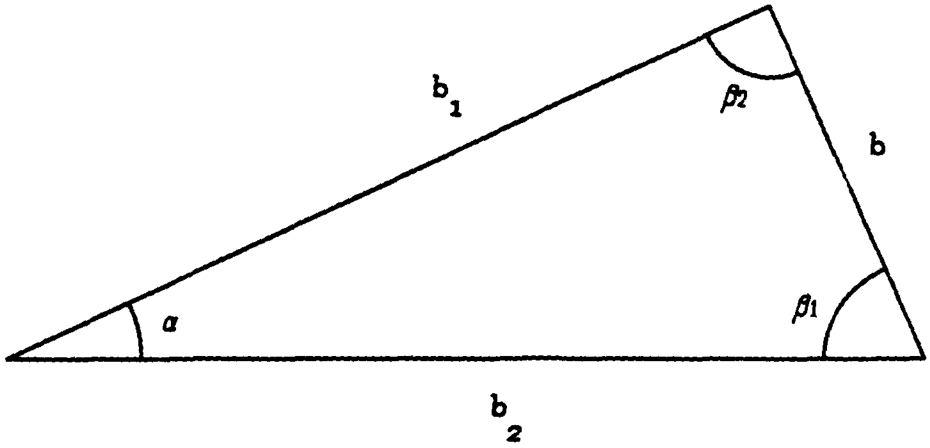


Figure 73. Minimizing $b/(b_1 + b_2)$

$$\frac{b}{b_1+b_2}=\frac{b}{\left(\frac{\sin(\beta_1)b}{\sin(\alpha)}+\frac{\sin(\beta_2)b}{\sin(\alpha)}\right)}$$

$$=\frac{1}{\left(\frac{\sin(\beta_1)+\sin(\beta_2)}{\sin(\alpha)}\right)}$$

$$=\frac{\sin(\alpha)}{\sin(\beta_1)+\sin(\beta_2)}$$

This form is more appropriate since α is fixed and we wish to find the minimizing values for β_1 and β_2. Since α is fixed, $\frac{\sin(\alpha)}{\sin(\beta_1)+\sin(\beta_2)}$ is minimized when $\sin(\beta_1)+\sin(\beta_2)$ is maximized. Arbitrarily, assume β_1 is greater than or equal to β_2 and find β_2 so that $\sin(\beta_1)+\sin(\beta_2)$ is maximized. Again, by the law of sines,

$$\sin(\beta_2)=\frac{b_2\sin(\beta_1)}{b_1}$$

Substituting,

$$\sin(\beta_1)+\frac{b_2\sin(\beta_1)}{b_1}$$

$$=\sin(\beta_1)(1+b_2/b_1)$$

since $\beta_1 \geqslant \beta_2$ then $b_1 \geqslant b_2$. Therefore,

$$\sin(\beta_1)(1+b_2/b_1)$$

is maximized when $b_1=b_2$.

We now consider the ratio of $\dfrac{b}{b_1+b_2}$ when $b_1=b_2$.

By the law of cosines

$$b^2=b_1{}^2+b_2{}^2-2b_1b_2\cos(\alpha)$$

Substituting equals

$$=2b_1{}^2-2b_1{}^2\cos(\alpha)$$

$$=2b_1{}^2(1-\cos(\alpha))$$

The original fraction squared is

$$\frac{b^2}{4b_1{}^2}=\frac{2b_1{}^2(1-\cos(\alpha))}{4b_1{}^2}$$

$$\frac{b}{2b_1}=\left(\frac{2b_1{}^2(1-\cos(\alpha))}{4b_1{}^2}\right)^{1/2}$$

$$=\left(\frac{1-\cos(\alpha)}{2}\right)^{1/2}$$

which by identity is $\sin(\alpha/2)$.

Therefore

$$\frac{b}{2b_1}=\sin(\alpha/2)$$

and

$$\frac{b}{b_1+b_2}\geqslant\frac{b}{2b_1}=\sin(\alpha/2)$$

Therefore, when $\sin(\alpha/2)>u_1/u_2$ no optimal-cost path through the region exists. *QED.*

We note that all the conditions expressed in Lemma 11 must be met for the result to hold. If some angle between sides B_1 and B_2 is non-convex, then an optimal-cost path may include an endpoint of either of the two sides. Figure 74 exemplifies such a case. Here, the angle formed by sides B_1 and B_2 meets the criterion of Lemma 11. However, vertex P is a non-convex vertex that is formed by two sides of the high-cost region that are between B_1 and B_2. Lemma 11 cannot be applied in this case. An optimal-cost S-to-G could easily involve vertex P, as illustrated in Figure 74.

Thus, a more global view of a homogeneous-cost region can invalidate the localized general nature of Lemma 11. However, taking a global view of the problem can also lead to additional pruning criteria. Recall that wedges terminate when their left and right boundary-defining paths intersect the bounding box. If a wedge-defining path travels directly from the wedge tip to a side of the bounding box at optimal cost, then there is an opportunity to prune the wedge. If the conditions expressed in Lemma 12 hold, the wedge can be discarded.

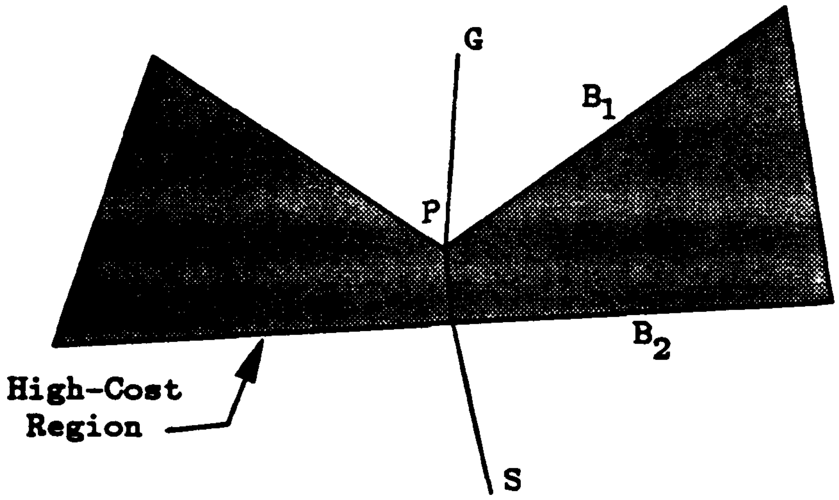


Figure 74. Lemma 11 Does Not Apply When All Angles of the High-Cost Region Are Not Convex Angles

Lemma 12: If R_L is a Snell's-law path defining the left boundary of wedge W, R_L travels from the wedge tip to a side of the bounding box at optimal cost and the goal, G, lies to the left of R_L, then the optimal-cost start-to-goal path does not lie inside wedge W. A similar result holds when G lies to the right of R_R, defining the right boundary of W, and R_R travels from the wedge tip to a side of the bounding box at optimal cost.

Proof: This situation is depicted in Figure 75 where the shaded triangles represent high-cost regions. Note that R_L travels from P, the wedge tip, to point B, on the bounding box, at optimal cost. The optimal-cost path to G involving vertex P includes a P-to-G path segment. Any path to G based on refinement of W must include the approach path to P. Thus, any path based on refinement of W must include a subpath between P and G. This subpath must have greater cost than the straight-line P-to-G path segment. Therefore, wedge W cannot contain the optimal-cost S-to-G path and may be pruned. *QED.*

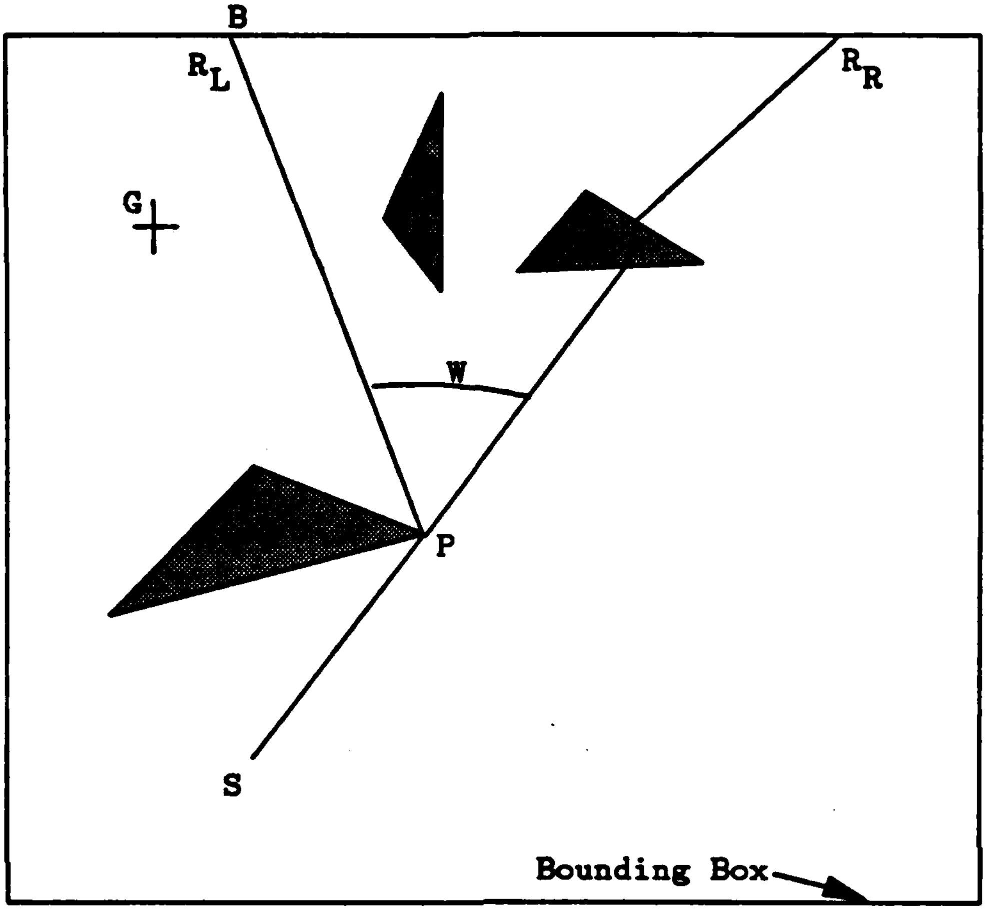


Figure 75. Pruning By Lemma 12

Note that Lemma 12 requires the strong condition that the wedge-defining path closest to the goal travel directly from the wedge tip to the bounding box at optimal cost. If this condition is not met (i.e., there are boundary-crossing episodes between the wedge tip and the path intersection with the bounding box), the wedge may contain an optimal-cost solution path, (as depicted in Figure 76) and Lemma 12 cannot be applied. Here, even though wedge W lies entirely to the right of goal, G, it still contains the optimal-cost solution path through vertex V.

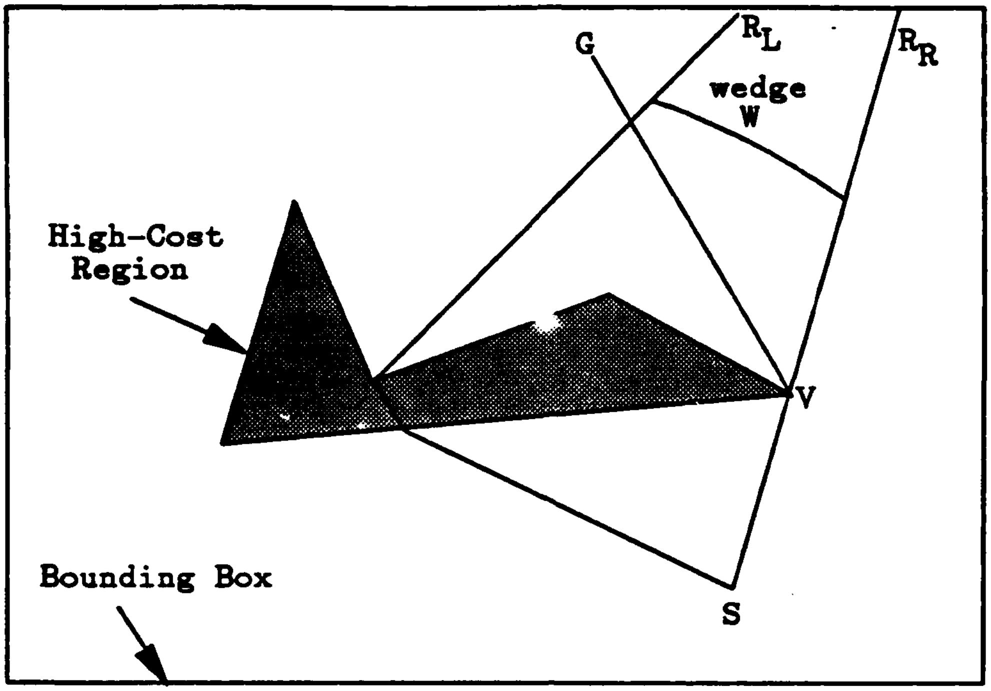


Figure 76. Lemma 12 Does Not Apply When the Wedge-Defining Path Closest to the Goal Does Not Travel from the Wedge Tip to the Bounding Box at Optimal Cost

Another opportunity to prune wedges relies on the global nature of the weighted-region problem. Informed strategies use knowledge of the location of the goal point to construct lower-bound evaluations for possible solutions. This strategy can also be applied to the weighted-region problem. Clearly, one lower-bound evaluation on the cost of a start-to-goal path is the Euclidean distance between the two points traveled at optimal cost. Better (i.e., tighter) lower bounds can be achieved by exploiting cached knowledge of the area-cost map. The next section develops this idea.

H. COST BOUNDS

Simple lower-bound cost evaluations for the optimal-cost path between any two points on the area-cost map can be achieved by assuming that the distance along the path is only the straight-line distance between the two points and that the path only goes through low-cost regions. There are also other methods to obtain lower-bound cost evaluations. Suppose that the globally optimal-cost solution path between every pair of region vertices (including obstacle region vertices) is precomputed and stored. These stored path costs can be used to construct lower-bound cost evaluations for the cost of the optimal path between any two points on the area-cost map.

Such a lower-bound cost evaluation is simply achieved by first locating two region vertices, V_G, the vertex closest to the goal, G, and V_S, the vertex closest to the start, S. Call the cached cost, C_P, the cost of the optimal V_S-to-V_G path. Let the cost of the straight-line path, traveled at the cost rate for point G, from G to V_G be C_G. Let C_S be the cost of the straight-line path from S to V_S, traveled at the cost rate for point S. Then, a lower bound on the cost of the optimal S-to-G path is:

$$C_P-(C_S+C_G)$$

Clearly, this must be a lower bound on the cost of the S-to-G path. Otherwise, the known-optimal cost of the V_S-to-V_G path is greater than the cost of the path from V_S to S to G to V_G. Similarly, an upper bound on the cost of the S-to-G path is:

$$C_P+C_S+C_G$$

This figure is an upper bound because the S-to-V_S and G-to-V_G path segments must be feasible when V_S is the closest vertex to S and V_G is the closest vertex to G.

Thus, stored-cost information can be used to construct both lower- and upper-bound evaluations for the cost of the optimal path between any two points on the area-cost map. Simply using Euclidean distance at optimal cost also yields a lower-bound point-to-point cost evaluation. However, the Euclidean distance path may not yield an upper bound if the path is not feasible. (If the straight-line path intersects obstacle areas, it is not feasible.) An upper bound, not relying on stored information, is achieved by finding the shortest-distance feasible path between start and goal and computing the actual cost of this path. Note that any feasible start-to-goal path acts as an upper bound on the cost of the optimal solution. Using the shortest-distance feasible solution is just a simple one.

Reliance on stored information seems to provide a method of achieving cost bounds quickly. However, if the space needed to retain the stored-path information is large, the caching may not be justified. If there are N vertices in the area-cost map then storing the cost of the optimal-cost path between each pair of vertices requires saving the combination of N path costs taken 2 at a time, N!/2(N-2)!. The storage requirement can be reduced (in the average case) by treating the region vertices as nodes in a graph. Links are entered in the graph between each pair of nodes (i.e., region vertices) that are connected by an optimal-cost path that does not include any other nodes. That is, if the optimal-cost path between two region vertices goes through some other region vertex, that cost need not be explicitly stored. If the optimal-cost path between region vertices only intersects boundaries of high-cost regions (at places other than endpoints) or the optimal-cost path is a straight-line segment, then the cost of that path must be stored. Clearly, given such a set of nodes and links, the cost of the optimal path between any two region vertices can be constructed by a standard graph search algorithm. Using such a graph to store path costs reduces storage requirements in the average case. In the worst case, the space requirement is still exponential.

We have seen that both upper- and lower-bound cost evaluations can be constructed for any point-to-point path. From Lemma 4, we can also find the minimum-cost path from the wedge tip to the Kth boundary of a K-explored wedge. In general, this information is more valuable in rating and pruning wedges than point-to-point evaluations based on the wedge tip and the goal. The results obtained by applying Lemma 4 to a K-explored wedge are more meaningful. However, Lemma 4 provides only a partial estimate; path cost from the Kth boundary to the goal must also be included so that a total point-to-goal lower-bound cost evaluation can be constructed. This requires that a line segment-to-point evaluation, not a point-to-point evaluation to be added to the cost of the minimal-cost path from the start to the Kth boundary of the K-explored wedge. Here, a simple evaluation assumes that there is a straight-line path from the line segment to the point (i.e., the goal) and that this path accrues cost at the optimal cost rate. To use this estimate, the distance from the line segment to the goal must be computed since it is the distance traveled by the straight-line path. The distance from a line segment to a point is the length of a normal to the line through the point if that normal segment intersects the original line segment.

Otherwise, the segment-to-point distance is taken as the minimum of the point-to-point distance from the two line segment endpoints to the point.

Stored costs can also be used to compute line segment-to-point lower-bound cost evaluations. The following two lemmas establish both the proof and the methodology.

Lemma 13: (Refer to Figure 77.) If a K-explored wedge contains both endpoints, P1 and P2, of the Kth boundary, a lower-bound evaluation from P1 to goal G is C1, a lower-bound evaluation from P2 to G is C2, the distance from P1 to P2 is D, and the optimal cost rate on the area-cost map is C_O, then a lower-bound cost evaluation from boundary K to G is:

$$\frac{C1+C2-DC_O}{2}$$

Proof: (Refer to Figure 77.) For each point X on the Kth boundary, two lower-bound cost evaluations are available, one based on P1 and one based on P2. That is, if C_{X1} is the cost of the X to P1 path and C_{X2} is the cost of the X to P2 path (both are based on the distance from X to the endpoint traveled at optimal cost), then C1 - C_{X1} and C2 - C_{X2} are both lower-bound cost evaluations for a path from X to the goal G. Since these are both lower bounds, the greater of the two costs is a tighter lower bound. However, there could be some other point X' also on boundary K that is less expensive to connect to G. That is, a path through X' has a lower maximum lower-bound evaluation than does the path through point X. To construct a lower bound on the cost for any path between boundary K and G, we must find X' so that it is the point on boundary K that has the minimum maximum lower-bound evaluation. Then, every other point on K will have one

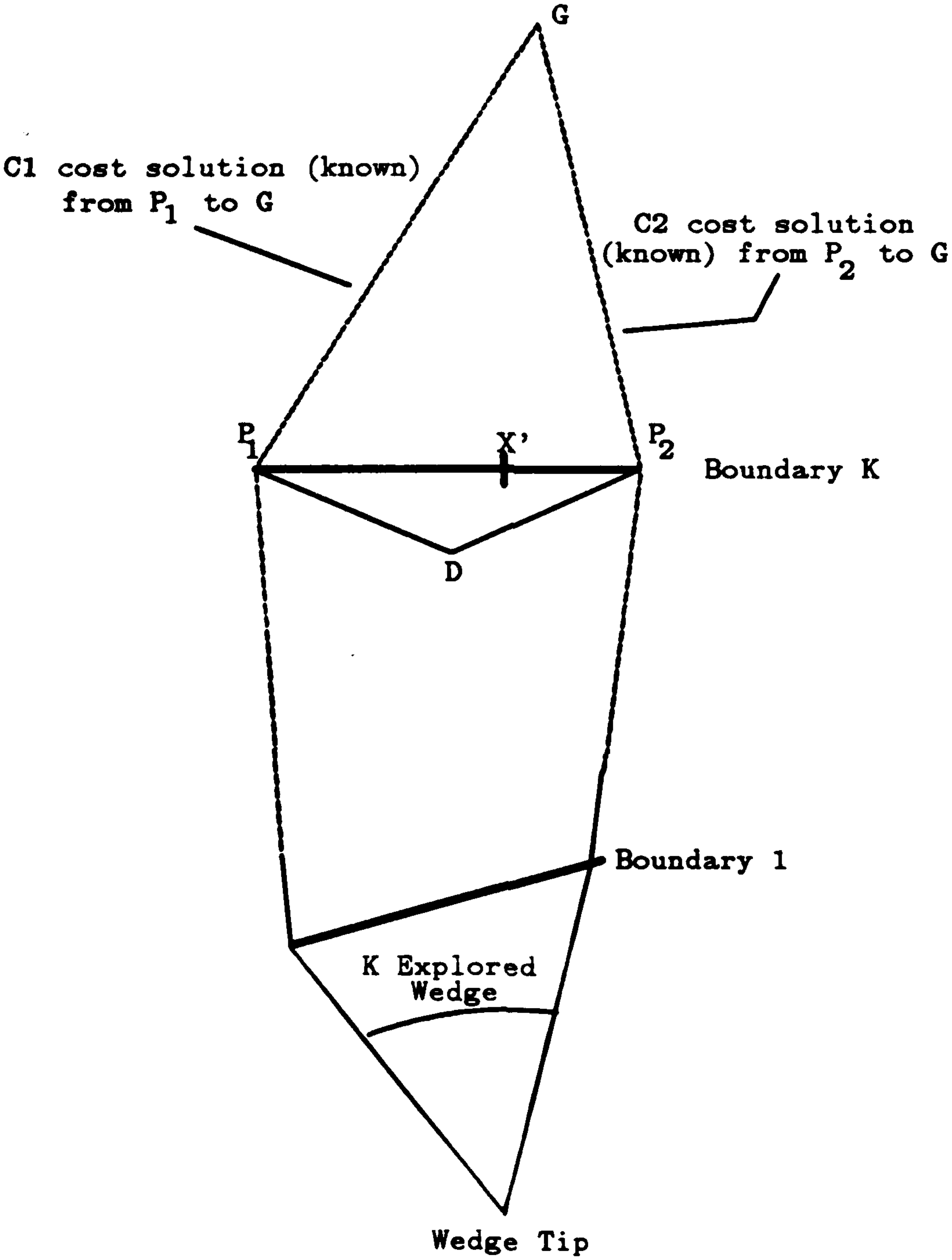

G
C1 cost solution (known) from P_1 to G
C2 cost solution (known) from P_2 to G
P_1
X'
P_2
Boundary K
D
Boundary 1
K Explored Wedge
Wedge Tip

Figure 77. Example for Lemma 13, A Wedge Contains Both Endpoints of Boundary K

lower-bound evaluation that is greater than the maximum lower-bound evaluation through point X'. The minimum maximum lower-bound cost evaluation occurs at the point X' when the two evaluations for that point are equal (i.e., the evaluation based on a path through point P1 equals the evaluation based on the path through point P2). This is true since, as the location of X' moves away from P1 towards P2, C_{X1} decreases monotonically while C_{X2} increases monotonically. Thus, in general, as X' moves away from P1 to P2, the maximum lower-bound evaluation for X' is based on the path through P1 until the evaluation based on a path through P2 overtakes it. From this point on, the maximum lower bound for X' is based on the path through P2 and this bound increases. Thus, the minimum maximum lower bound for X' occurs when the evaluation based on the path through P1 equals the evaluation based on the path through P2. Let D be the distance between P1 and P2 and let D_X be the distance between P1 and X'. Then the two evaluations for X' are equal when:

$$C_1 - D_X C_O = C_2 - (D - D_X) C_O$$

$$C_1 - D_X C_O = C_2 - DC_O + D_X C_O$$

$$C_1 = C_2 - DC_O + D_X C_O + D_X C_O$$

$$C_1 = C_2 + 2D_X C_O - DC_O$$

$$C_1 - C_2 + DC_O = 2D_X C_O$$

$$\frac{C_1 - C_2 + DC_O}{2} = D_X C_O$$

Now note that the lower-bound cost evaluation for a point D_X distance away from P1 on K is:

$$C_1 - D_X C_O$$

$$=C_1-\frac{C_1-C_2+DC_O}{2}$$

$$=\frac{2C_1}{2}-\frac{C_1-C_2+DC_O}{2}$$

$$=\frac{2C_1-C_1+C_2-DC_O}{2}$$

$$=\frac{C_1+C_2-DC_O}{2}$$

QED.

Lemma 14: (Refer to Figure 78.) Let E_1 and E_2 be the points on boundary K where the two wedge-defining Snell's-law paths of a K-explored wedge intersect boundary K. Let P_1 and P_2 be the endpoints of boundary K, C_1 be the lower-bound cost evaluation from P_1 to the goal, G, and C_2 be the lower-bound cost evaluation from P_2 to G. Assume $C_1>C_2$. Let D_1 be the distance from E_1 to P_1 assumed to be traveled at the optimal cost on the area-cost map, C_O. Let D_2 be the distance from P_2 to E_2, also assumed to be traveled at cost C_O. Let D be the distance between P_1 and P_2. If

$$D_1<\frac{C_1+C_2-DC_O}{2}<D_2$$

then the lower-bound evaluation from boundary K to G is

$$\frac{C_1+C_2-DC_O}{2}$$

If $D_1>\dfrac{C_1+C_2-DC_O}{2}$ then the lower-bound evaluation is

$$C_1-D_1$$

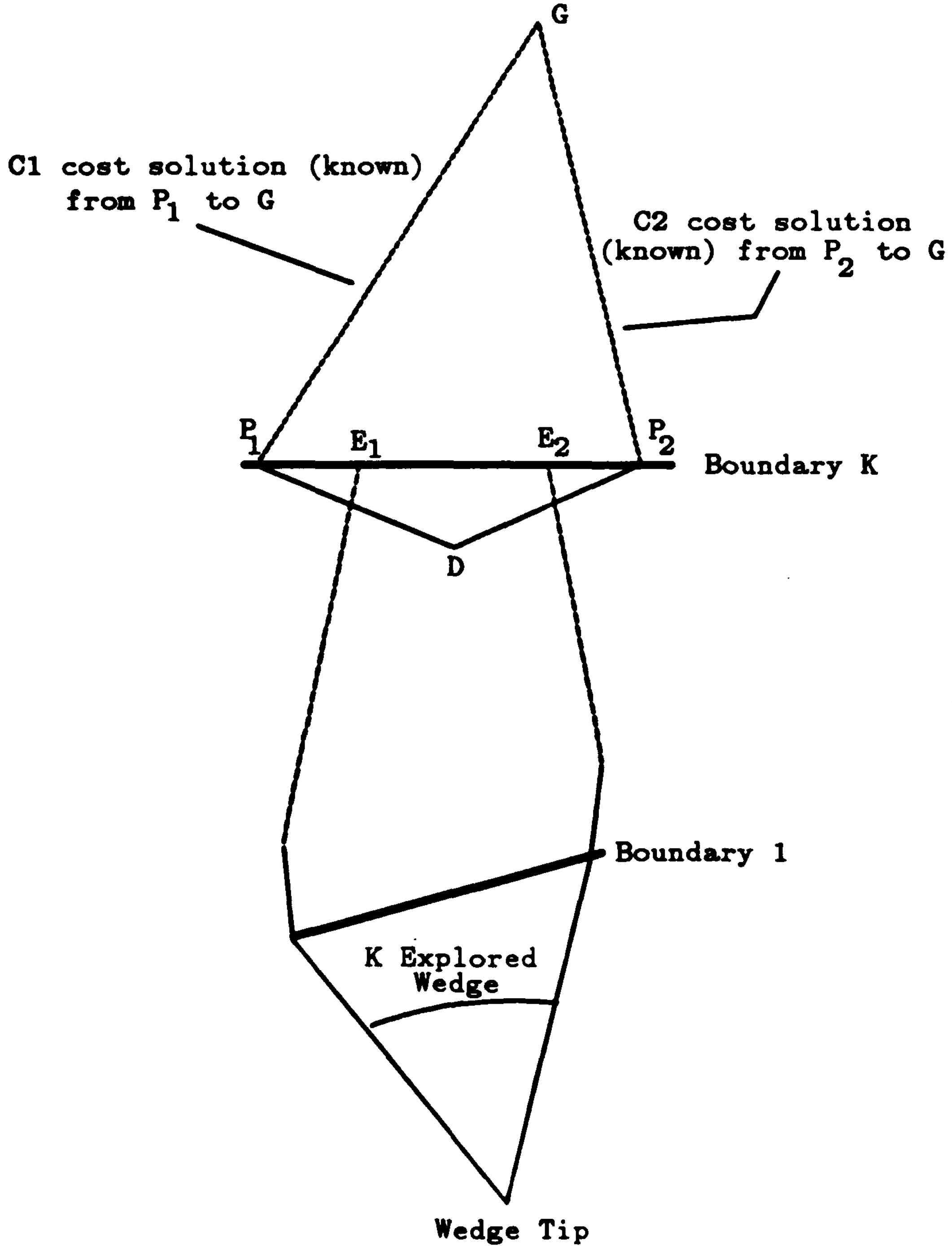

G
C1 cost solution (known) from P_1 to G
C2 cost solution (known) from P_2 to G
P_1
E_1
E_2
P_2
Boundary K
D
Boundary 1
K Explored Wedge
Wedge Tip

Figure 78. Example for Lemma 14, A K-explored Wedge Does Not Contain the Endpoints of Boundary K

Otherwise, $D_2 < \dfrac{C_1+C_2-DC_O}{2}$ and the lower-bound cost evaluation is

$$C_1 - D_2$$

Note that the lower-bound cost evaluations are based on the greater of C_1 and C_2. If $C_2 > C_1$ then the evaluations are based on C_2.

Proof: Lemma 13 is a special case of Lemma 14 and the proofs are similar. The problem is to find the point on K, within the wedge, that has the minimum maximum lower-bound evaluation. This is given according to the construction in Lemma 13. In case 1 of Lemma 14, the point X' which yields the minimum maximum lower-bound evaluation for segment K lies in the interval between E_1 and E_2, and is thus the appropriate evaluation for the E_1-E_2 segment. In the second case of Lemma 14, X' falls in the interval between P_1 and E_1, and E_1 is the closest that any point in the E_1 to E_2 interval can get to X'. Since maximum lower-bound evaluations are monotonically increasing on both sides of X', the evaluation based on a path through point E_1 is the best one available for the E_1 to E_2 interval. This evaluation is exactly $C_1 - D_1$. In the final case of Lemma 14, the point X' falls between E_2 and P_2. Here, an argument similar to that for the second case above produces an evaluation based on the path from E_2 through P_2 to G. This evaluation is $C_1 - D_2$. *QED.*

I. PHYSICAL BOUNDS

In Chapter III, the concept of using an ellipse to physically constrain the size of the search graph was introduced. The same idea can be incorporated into a scheme relying on a homogeneous-cost-region problem representation. Instead of limiting the search to those nodes within the ellipse, the region-based scheme examines only those regions (or portions of regions) that are inside the ellipse. We now formally prove that an ellipse can be constructed so that it must contain the optimal-cost solution path for a weighted-region problem.

Lemma 15: Given a cost C_F of a feasible start (S) to goal (G) solution path, then an ellipse having foci at S and G constructed so that for each point P_E on the ellipse boundary, the sum of the distance from P_E to G plus the distance from P_E to S multiplied by the optimal cost C_O is equal to C_F must contain the optimal-cost solution path.

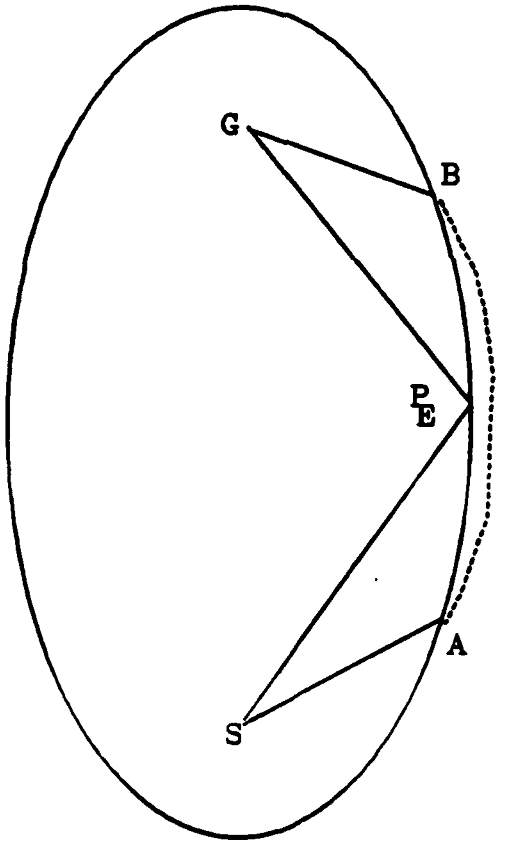


Figure 79. Limiting Ellipse

Proof: (Refer to Figure 79.) If the ellipse entirely contains the optimal-cost solution path, the proof is done. Thus, assume that some portion of the optimal-cost solution path exits the ellipse at a point A. Because the ellipse circumscribes the goal G, any start-to-goal solution path must again enter the ellipse at some other point B. The shortest-distance connection between A and B that does not enter the ellipse between A and B travels right along, but just outside the ellipse boundary (similar to the dashed line in Figure 79). Assume that the connection between A and B takes such a course and that it can be traveled at cost C_O. Note that, by construction, the segments S-P_E and G-P_E are as if traveled at optimal cost. Also, assume S-A and G-B can be traveled at cost C_O. Assuming that S-A, G-B and A-B can be traveled at C_O produces the least-cost path that does not lie entirely inside the ellipse. Also, these assumptions factor cost out of the problem. Clearly, the S-P_E-G path has less distance than the S-A-B-G path. Therefore, the optimal-cost path must lie entirely within the constructed ellipse. Note that, by construction, the feasible solution having cost C_F must also lie within the ellipse. Therefore, there is at least one feasible solution within the ellipse. *QED*.

The ellipse can also be made smaller iteratively. Once the ellipse based on C_F is constructed, there must be some lowest cost rate associated with only those homogeneous-cost regions inside the ellipse. The ellipse can be reconstructed if this inside lowest cost does not equal C_O. In fact, the reconstruction can continue until the ellipse contains a homogeneous-cost region having cost rate equal to the (lowest) cost rate used in the construction of the ellipse. A proof is very similar to that given for Lemma 15 and is not repeated.

The bounding box is constructed to be a rectangular figure that entirely contains the limiting ellipse of Lemma 15. The width of the bounding box is equal to the ellipse minor axis and the rectangle height is equal to the ellipse major axis. The bounding box is placed on the area-cost map so that each of its sides is tangent to the ellipse at exactly one point (see Figure 80). Clearly, the bounding box must entirely contain the optimal-cost solution path since it contains the limiting ellipse of Lemma 15.

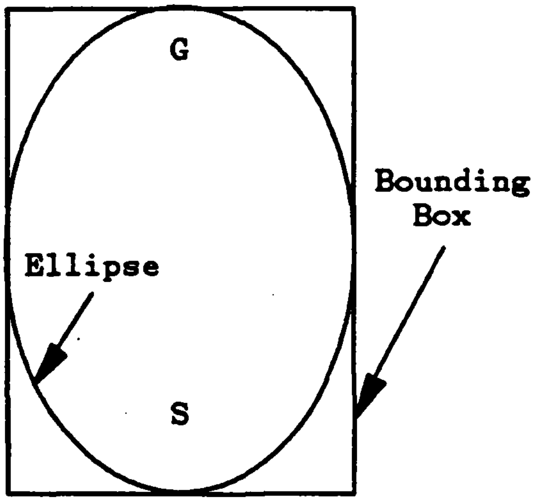


Figure 80. Bounding Box Circumscribes Ellipse

The feasible solution required to construct the limiting ellipse of Lemma 15 can be achieved by any of the point-to-point upper-bound methods previously discussed. There is also a method to construct a feasible solution based on stored information. Suppose we construct the straight-line path between the start and goal and locate the boundary-crossing episode on the path closest to the goal E_G and the boundary-crossing episode closest to the start E_S. Let E_{G1} and E_{G2} be the endpoints of the boundary on which E_G is located. Similarly define E_{S1} and E_{S2}

relative to E_S. If the paths between region vertices are stored, we can find the least-cost path between E_G and E_S based on paths through boundary endpoints. The paths S-E_S and G-E_G must be feasible and can be connected by the least-cost E_S-E_G path, resulting in a feasible S-to-G path.

In addition to cached path costs, cached path locations can also be used to physically limit the search space. In Figure 81, known optimal-cost paths between P_1 and P_2 and between P_3 and P_4 both originate and terminate outside the bounding box. Further, both S and G are inside the polygon defined by these two paths and the bounding box. By Lemmas 10 and 13, the optimal-cost S-to-G path must lie entirely between the P_1-to-P_2 path and the P_3-to-P_4 path. Thus, cached path locations can also be used to limit the search space in the weighted-region problem. Using cached information in this manner amounts to a subtle form of learning. Exploitation of such a primitive learning capability is hampered by indexing problems. It is difficult (and thus time consuming) to select the best pair of stored paths in order to maximally limit the search space. However, occasions arise when the search space can be severely constrained, as exemplified by Figure 82. Here, the problem neatly decomposes into finding the least-cost S-to-A path, B-to-G path, and then connecting these two paths by the stored A-to-B approach path. We also note that "learning by doing" [Ref. 45] could be exploited in a similar manner. Solutions to previously solved instances of weighted-region problems could be stored, allowing the accumulation of a base of knowledge over time.

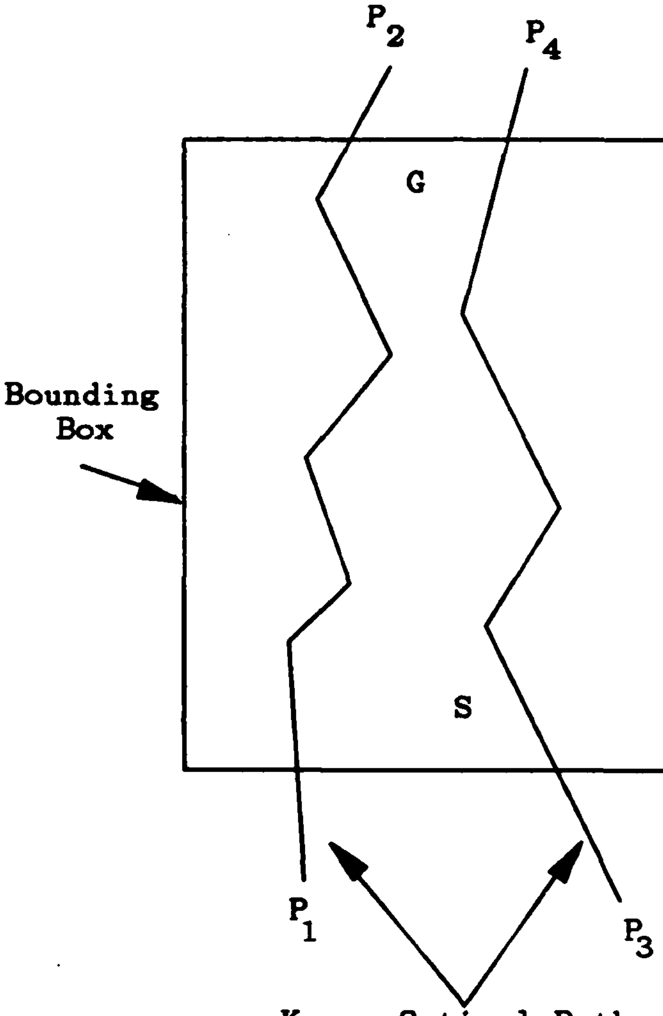


Figure 81. Constraint By Known Paths

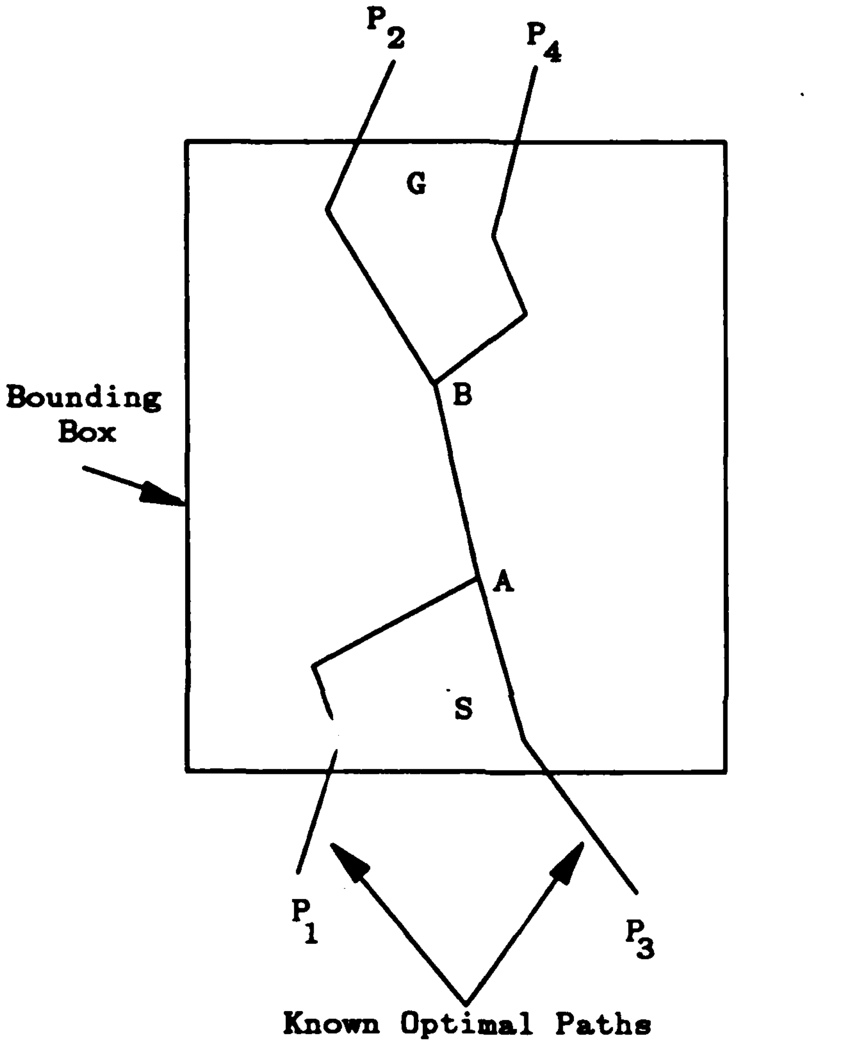


Figure 82. Highly Constrained Search Space

J. LIMITATIONS IN THE APPLICATION OF SNELL'S LAW

Section IV.C clearly shows the applicability of Snell's law to the weighted-region problem and the following sections develop properties of the law when so applied. There are, however, characteristics of the law that are incompatible with the weighted-region problem. The first of these is that Snell's law does not apply to obstacle areas. Even when obstacles are modeled as regions of infinite cost rate, Snell's law will still allow paths to go through them. There is a trivial solution to the obstacle problem. We have defined a Snell's-law path so that it ends when it intersects the bounding box. We extend the definition so that Snell's-law paths also terminate if they intersect an obstacle region boundary.

A second problem involves the total internal reflections allowed by Snell's law. Recall the equality relation expressed by Snell's law:

$$\sin(\theta_1)/U_1 = \sin(\theta_2)/U_2$$

Here, U_1 and U_2 are inverses of refractive indices, θ_1 is an angle of incidence and θ_2 is an angle of refraction. (When applied to the weighted-region problem, U_1 and U_2 become cost-rate reciprocals.) There is no known closed-form solution yielding the Snell's-law path between two given points. However, given an initial point and heading, a Snell's-law path can be constructed. Finding the Snell's-law path between two given points relies on an iterative ray-tracing operation. Snell's-law paths are iteratively constructed until the desired points are connected (within a given tolerance in that the Snell's-law path begins at one point and passes within some tolerance of the second point).

Constructing a Snell's-law path given an initial point and heading requires directly solving for the angle of incidence θ_1 at each boundary-crossing episode on the path. Once θ_1 is known, the transformed Snell's-law equation,

$$U_2 \sin(\theta_1)/U_1 = \sin(\theta_2)$$

is used to solve for the sine of the angle of refraction (corresponding to the same boundary-crossing episode). Dependent upon the values in the transformed equation, the quantity $U_2 \sin(\theta_1)/U_1$ can be greater than 1, meaning a total internal reflection. For this to occur, the ratio U_2/U_1 must be greater than unity. Thus, in the weighted-region problem, total internal reflections only occur when a path exits a high-cost region, entering a low-cost region. In this case, a path that obeys Snell's law "bounces off" the region boundary, back into the high-cost region at a new heading of $((U_2 \sin(\theta_1))/U_1)-1$. Clearly, a path that "doubles back on itself" in this fashion makes no sense in the weighted-region problem. An algorithm that relies on Snell's law as a guiding principle must make provisions for internal reflection paths.

In the weighted-region problem, the only interesting reflection paths are those that occur when $((U_2 \sin(\theta_1))/U_1)=1$ because these are the only reflection paths that do not "double back" into high-cost regions. In this case, θ_1 is called the *critical angle* and $\theta_1 = \mathit{arcsin}(U_1/U_2)$. Such paths often provide the optimal-cost path between a start inside a high-cost region and a vertex of that region. as illustrated in Figure 83. Here, the optimal-cost path from S to V intersects side B_1 of the high-cost region at the critical angle. In this case, we say that the path *critically uses* side B_1 of the high-cost region.

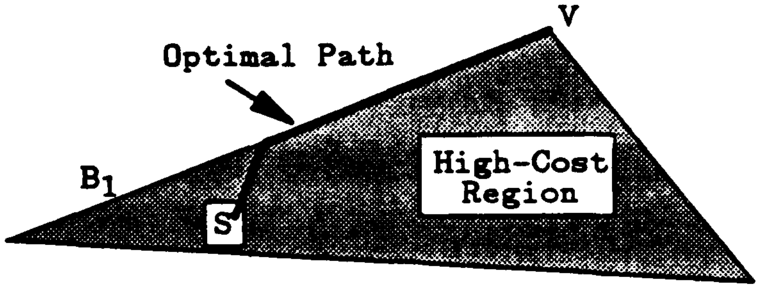


Figure 83. A Reflection Solution Path

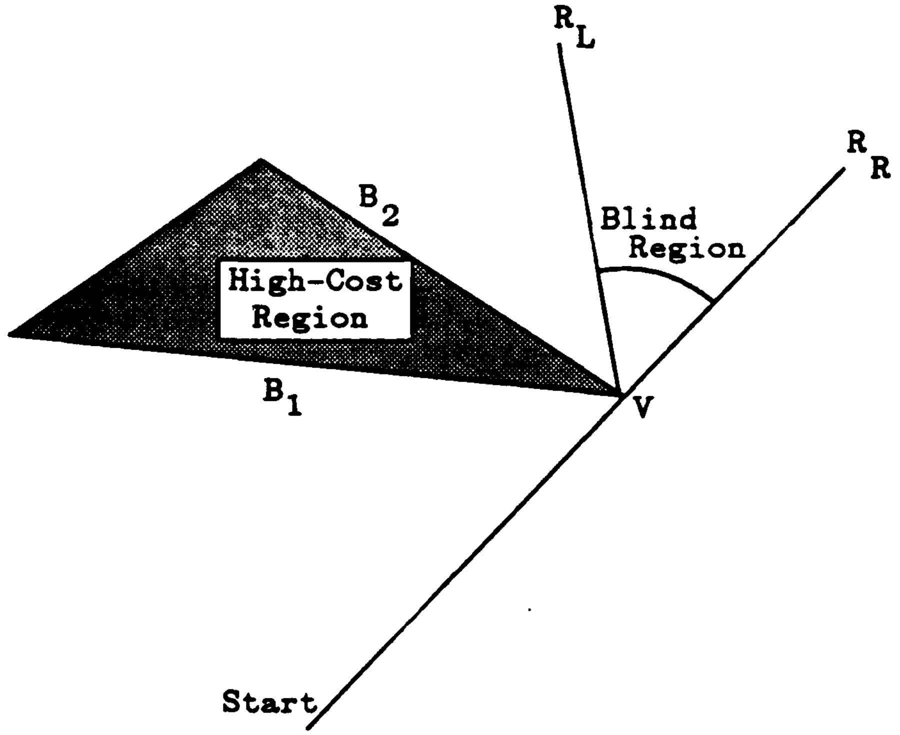


Figure 84. A Blind Region

A third problem area arises due to the existence of *blind regions*. These regions correspond to portions of the area-cost map that cannot be reached by any Snell's-law path beginning at the start location. Figure 84 depicts such an instance. Here, path R_L is refracted at vertex V by intersections with sides B_1 and

B_2 of the high-cost region. Path R_R travels infinitesimally close to vertex V but it does not intersect any side of the region. Conceptually, R_L is the rightmost path through V that intersects the high-cost region and R_R is the leftmost path through V that does not intersect the region. Thus, any path to the right of R_L does not intersect the region and any path to the left of R_R does. By Theorem 1, any path to the right of R_L is also to the right of (or identical to) R_R and any path to the left of R_R is also to the left of (or identical to) R_L. Therefore, there is no Snell's-law path from the start to any point inside the wedge defined by R_L and R_R. By definition, such a wedge is a blind region. The optimal-cost path to any point within this blind region wedge is constructed by finding the optimal path from V to that point and appending that path to the predefined S-to-V approach path. Snell's law is not applied at vertex V where the two paths are joined (although Snell's law does apply in the limit when a vertex is modeled by a small curve, not by the intersection of two lines).

There is an analogy from the field of optics that applies in cases where paths include a blind-region vertex as a turn point. The situation is similar to (single slit) diffraction optics (however, in the path-planning problem, the path is constrained to remain within the boundaries of the wedge). Let a vertex at the base of a blind region be denoted as a *diffraction vertex*. Thus, vertex V of Figure 84 is a diffraction vertex.

When blind regions occur, they represent an opportunity for the start-to-goal path-planning problem to be recursively decomposed. Any optimal-cost path to a point inside the blind region must include the approach path from the start to the diffraction vertex. The path-planning problem from the diffraction vertex to the goal is a subproblem and the solution path for this subproblem must be within the wedge defining the blind region. If a diffraction vertex-to-goal solution path is found, it can be appended to the stored approach path, resulting in a start-to-goal solution path.

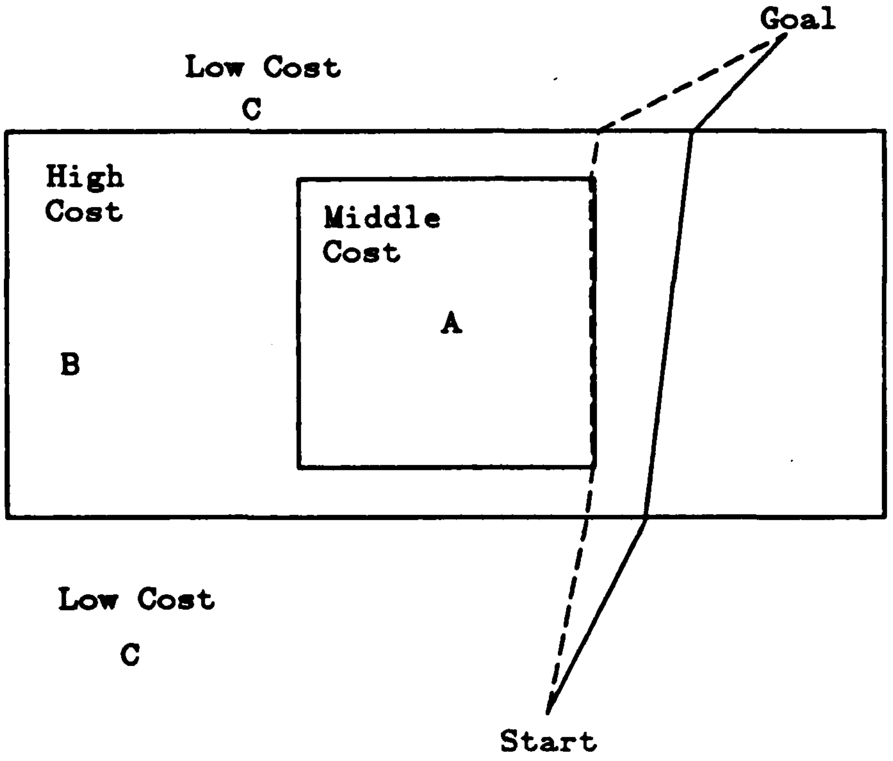


Figure 85. Locality Aspect of Snell's Law

A final difficulty with Snell's law stems from the totally localized nature of the criteria. Consider a situation similar to that depicted by Figure 85. In terms of cost, let region C be the most favorable, region A the next best and region B be the least-favorable region. Snell's law simply perturbs the original straight-line start-to-goal path until an optimal-cost path involving regions C and B is determined (as represented by the solid line). The law has no ability to consider alternate paths involving region A, even though the optimal-cost path could easily be similar to the dashed line path of Figure 85. Thus, Snell's law implicitly relies on proposing a straight Start-to-Goal line as an initial path. The law ignores more favorable adjacent areas unless either the initial straight-line path intersects them or the optimization procedure accidentally perturbs the path into an intersection with them.

This situation can lead to a form of the same problem that affects the wavefront-propagation technique, i.e., combinatorial explosion and computational excess. Solving the weighted-region problem by applying Snell's law to a large number of distinct regions requires that each region vertex within some circumscribing limit be specifically examined. The problem is somewhat less serious than in wavefront propagation because the size of each individual area requiring examination is generally much larger (and thus there are fewer region vertices in the same area) than the areas used in a lattice representation. However, if a path involving a significant distance from the start location to the goal location is required, the size of the areas has less of an ameliorating effect. Moreover, the type of technique we are attempting to develop must produce a solution efficiently to be valuable. It does not appear that a method which must "look everywhere" can fulfill this requirement. Extending the situation in Figure 85 to its absolute limit when many small distinct regions are present, the application of Snell's law is tantamount to using an increased-overhead wavefront-propagation technique.

Situations similar to those depicted by Figure 85 represent only one class of problems that must be addressed when applying Snell's law to instances of the weighted-region problem. Whenever a large number of distinct homogeneous-cost regions occupy the problem space, their juxtaposition as well as their superimposition create difficulties. Consider a situation similar to that depicted in Figure 86. Let the regions A through E become progressively less costly as they move left of the straight-line start-to-goal path. Such a circumstance can occur in the real world when a mountaintop lies immediately between the start and goal. Traveling directly over the crest encounters greater elevation change and thus requires greater effort and cost (minimizing time). As the path shifts to the side of the crest, steepness, effort, and cost decrease. The problem of finding the least-cost route here reduces to locating the best tradeoff point between increased lateral distance and decreased cost to move forward. We can develop simple mathematical criteria, such as depicted in Figure 87, to determine the optimal tradeoff point for these special cases.

In Figure 87, let U1, U2, U3 and U4 represent region cost rates. Let d1, d2, d3 and d4 denote Euclidean distances. Let A and B mark the start and goal locations. Let θ_1 and θ_2 be angular measures. The path of small dashes represents a known start-to-goal path. The large dashed lines indicate normals to the cost rate U2 region boundary. The heavy line represents a possible path through the adjacent region. Then solving the equations:

$$\sin(\theta_1) = U4/U2$$

$$\sin(\theta_2) = U3/U2$$

produces the minimum-cost path through the adjacent region.

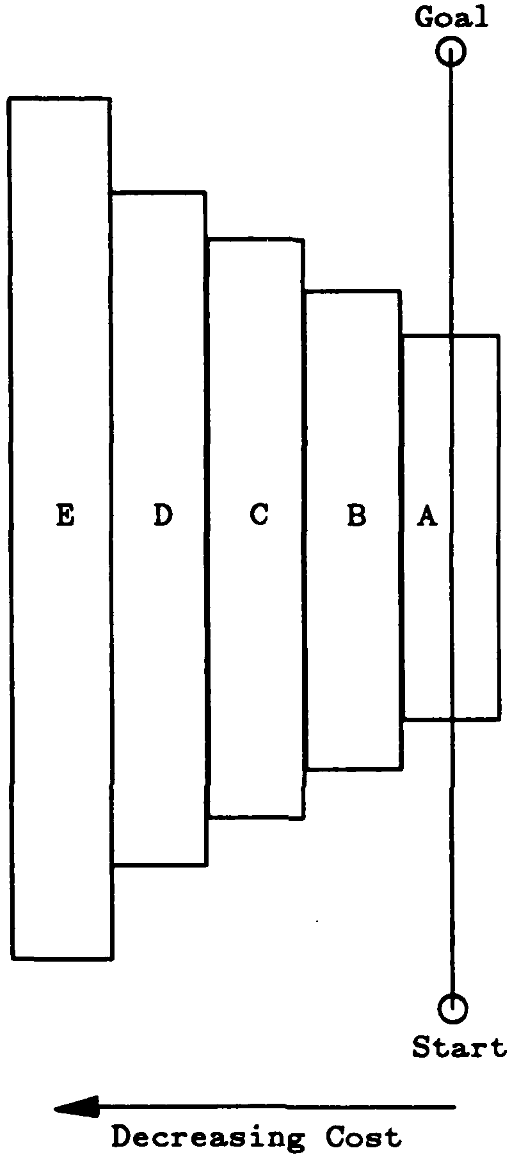


Figure 86. Adjacent Homogeneous-Cost Regions

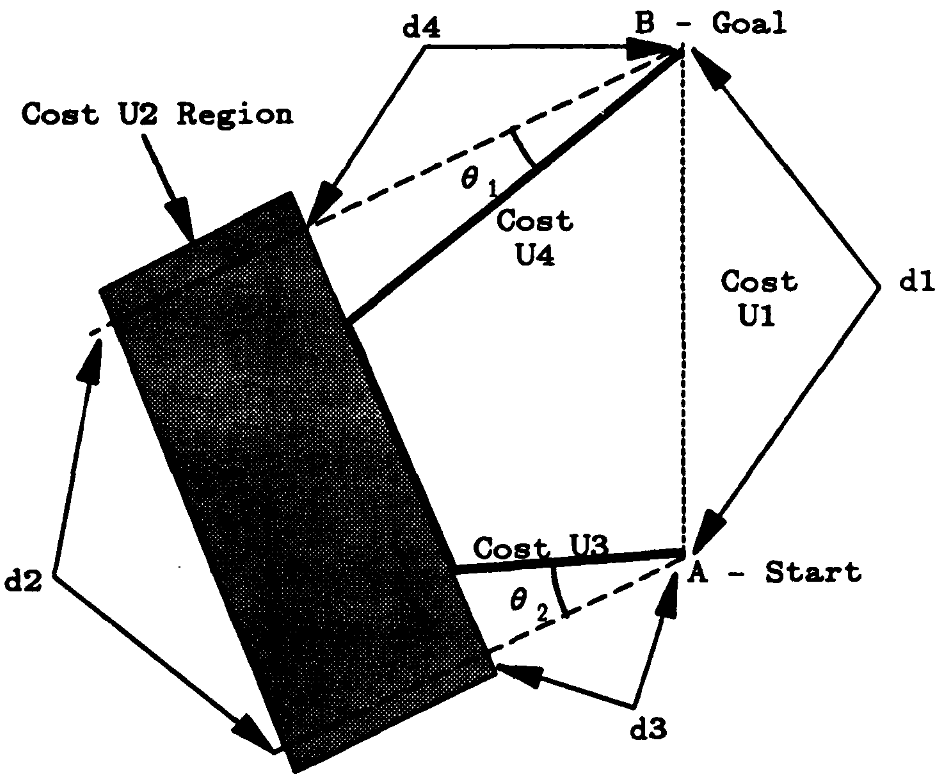


Figure 87. Optimal Paths Through Adjacent Areas

Substituting the sin(θ) values into the equation below determines the best path. If the inequality is true, the path through the adjacent area is best. Otherwise, the straight-line path is preferable. Note that the below formula can be used to specify how far away an adjacent area having a known cost rate can be located and still have the possibility of containing the optimal-cost path.

$$\frac{d1}{U1} > \frac{1}{U2}\left(d2 - \frac{d3\sin(\theta_1)}{\left(1-\sin^2(\theta_1)\right)^{1/2}} - \frac{d4\sin(\theta_1)}{\left(1-\sin^2(\theta_2)\right)^{1/2}} \right)$$

$$+\frac{1}{U3}\left(\frac{d3}{\left(1-\sin^2(\theta_2)\right)^{1/2}} \right) + \frac{1}{U4}\left(\frac{d4}{\left(1-\sin^2(\theta_1)\right)^{1/2}} \right)$$

The situation depicted in Figure 86 and the result developed in Figure 87 do not consider superimposed homogeneous-cost regions and both posit uniform-cost regions on the approach to each of the lateral regions (A, B, C, ...). The occurrence of such specialized cases in the real world is not likely. The main point is that it appears that the only way to determine the least-cost path in such real-world cases is to compute the actual cost of routes through each region (within some circumscribing limit) and make comparisons. Again, we have the "look everywhere" phenomena, the very problem that we are attempting to avoid.

K. SUMMARY

Snell's law can serve as the local optimization criterion of a solution technique for the weighted-region problem. The law is well-suited for application to homogeneous-cost region problem representations. Based on Snell's law, wedges within the search space can be created. There are several methods to eliminate a wedge from consideration in efforts to solve the weighted-region problem. Both upper and lower bounds on path cost are useful in the pruning process. Precomputed, cached information can be used in constructing bounds, however, such knowledge is not required.

Snell's law also has some characteristics that are not well-suited to the weighted-region problem. These include dealing with obstacles, total internal reflections, blind regions and the localized nature of the law itself. These aspects of Snell's law must be compensated for strategically, by a control algorithm. Developing such an algorithm that exploits the characteristics of Snell's law is the subject of the following chapter.

V. SNELL'S-LAW-BASED A* SEARCH

A. INTRODUCTION

Section IV.C derived Snell's law as a consequence of using partial derivatives to solve a minimum-cost path planning problem. The Continuous Dijkstra Algorithm (CDA) reported in [Ref. 42] establishes an algorithmic precedent for relying on Snell's law to solve the weighted-region problem. Thus, the applicability of the law is clear. Snell's law provides the basic guiding principle for a search strategy, as does the straight-line principle for binary-case techniques. Both are local optimality criteria. In this chapter, we develop a weighted-region-problem algorithm, based on Snell's law, that offers the potential for improved average-case performance (in both time and space) over all competing techniques.

Recall that Dijkstra's algorithm (and thus the continuous Dijkstra technique itself) is an uninformed strategy. The preceding chapter developed methods to evaluate upper and lower bounds on the cost of possible start-to-goal paths through specific wedges. The evaluations are based on a knowledge of the goal location. The cost bounds can also be used to rate wedges; the wedge having the lowest lower-bound cost evaluation should be rated as the most favorable wedge and be the first wedge refined. Thus, we have a method to order the search of a set of wedges, based on their likelihood of containing the optimal solution. By definition, we have the elements necessary to construct an agenda-based informed search strategy. A* search is the archetypal informed strategy and serves as the basis for the algorithm we present.

To utilize A* search, the homogeneous-cost region problem representation must be converted into a search graph. A similar technique is applied to binary-case problems by the successful VGraph algorithm. A salient difference is that the binary assumption is untenable in the general-case weighted-region problem. As a result, feasible turning points on solution paths are not limited to the members of a finite set of predefined region vertices. Thus, a finite predefined graph of region vertices will not provide a solution path for the general-case weighted-region problem. Our algorithm relies on Snell's law to dynamically create a graph of wedges that correspond to areal subdivisions of the physical environment. Recall that three sub-wedges can be created from a single wedge based on the Snell's-law path to the closest unsolved search point within that wedge. Thus, in general, the search graph that we construct has a branching factor of three.

At this point in the discussion it seems appropriate to define the search space for our problem. The start state has two initial wedges, a feasible start-to-goal solution path (which may or may not have optimal cost), and a lower-bound evaluation for each the two initial wedges. The lower-bound evaluation for a wedge is based on the cost of a possible start-to-goal path within that wedge. The single operator to transform states is wedge *refinement*, or the creation of sub-wedges based on the Snell's-law path to the closest unsolved search point within the wedge being refined. Thus, each search state in the search space is a refinement of its successor state and includes a wedge, the least-cost start-to-goal solution path found so far, and a lower-bound evaluation for that wedge. The goal state is reached when every state in the search space includes a lower-bound evaluation that is greater than the cost of the least-cost start-to-goal path that has been found.

As a basic strategy, we rely on Snell's law to create, refine, rate and search wedges corresponding to nodes in a graph. Upper and lower bounds on the cost of possible paths through wedges are used. Wedges are refined (and removed from the search space) in order of their lower-bound cost evaluations. If the evaluation for the wedge having the lowest lower-bound cost evaluation exceeds the upper bound, the search terminates. Figure 88 depicts a preliminary version of the control flow for the algorithm that we use for explanatory purposes. A final version of the algorithm control flow is provided at the end of this chapter. Much of the remainder of this chapter is devoted to explaining the actions required in each box of Figure 88.

We also note a subtle assumption that underlies our algorithm development. We assume a ternary-cost map (as defined in Section IV.B) since it is the simplest classification scheme that includes the difficulties associated with non-binary region descriptions. The algorithm we develop requires only minor modification to solve arbitrary weighted-region problems, including those that represent any number of different cost rates. We also note that a ternary scheme is appropriate for some important autonomous agents [Ref. 27.44]. Also, classifications based on local sensor equipment are often, at best, ternary in nature.

A final introductory remark concerns the figures used in this chapter. Most of them are not produced by actual data provided by a functioning program. These figures are intended for illustrative purposes only and have no need of exact

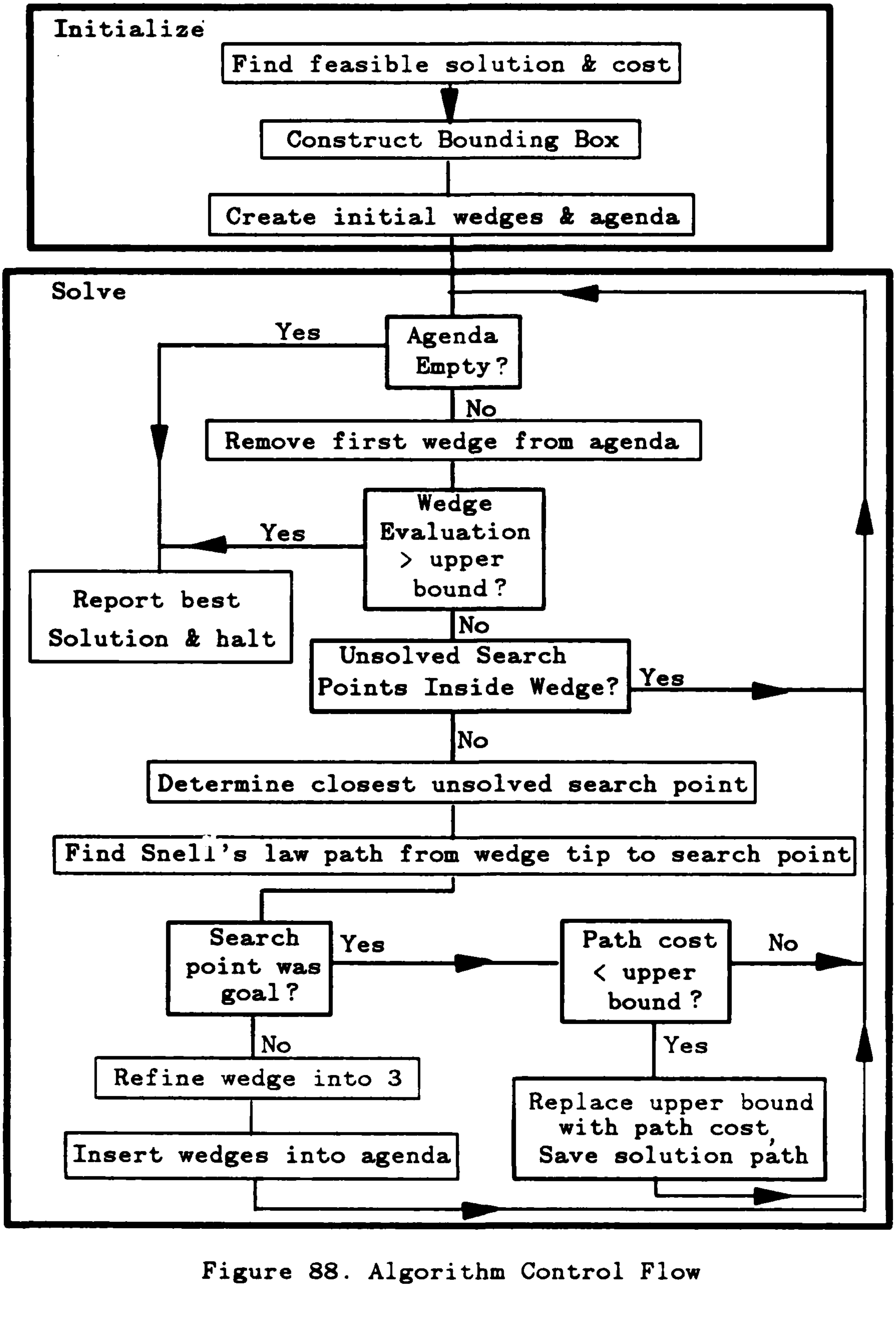

Initialize
Find feasible solution & cost
Construct Bounding Box
Create initial wedges & agenda
Solve
Agenda Empty?
Yes
No
Remove first wedge from agenda
Wedge Evaluation > upper bound?
Yes
Report best Solution & halt
No
Unsolved Search Points Inside Wedge?
Yes
No
Determine closest unsolved search point
Find Snell's law path from wedge tip to search point
Search point was goal?
Yes
Path cost < upper bound?
No
Yes
No
Refine wedge into 3
Insert wedges into agenda
Replace upper bound with path cost, Save solution path

Figure 88. Algorithm Control Flow

fidelity. (We note that Chapter VI and the Appendix contain many illustrations generated from an implemented Snell's-law-based algorithm as it solves different instances of the weighted-region problem.) There are some exceptions. Figures 95 through 99 in Section E are produced from tracing actual Snell's-law paths through high-cost regions. Also, Figure 122 is a (bit map) copy of a graphics screen produced by a working Snell's-law-based algorithm during execution. All other figures in this chapter are artificially created.

B. INITIALIZATION

Initialization must be performed for each specific weighted-region problem to be solved. The algorithm assumes access to an appropriate, ternary-cost map. Chapter IV presented two primary methods of obtaining an upper bound on the cost of an optimal start-to-goal solution path. In the absence of stored information, the problem can be treated as binary (ignoring cost regions but considering obstacles) to obtain a feasible solution. The actual cost of a feasible solution can be calculated on the ternary map. If stored information is available, an upper-bound point-to-point cost evaluation can be directly computed. Either method results in a satisfactory upper bound (on the cost of the optimal solution path) for initialization requirements. Let the cost of the feasible solution path (calculated on the ternary map) be U. Note that the cost U also serves as an initial globally-known upper bound on the cost of the optimal-cost solution path. This upper bound may be replaced during the solution process when a lower-cost feasible start-to-goal solution path is found.

Suppose that the optimal cost rate for the ternary-cost map is C_O (recall that this is a scalar cost per unit of distance traveled). We wish to construct an ellipse that physically contains the optimal-cost solution path and then circumscribe this ellipse by a rectangular bounding box. The major axis of the ellipse is equal to the rectangle height while the ellipse minor axis is the rectangle width. Thus, the major and minor axis of the ellipse must be computed, given U and C_O. (We note that a method of constructing the ellipse was discussed in the previous chapter. We now provide a construction mcthod that is more procedurally oriented and is suitable for constructing the rectangular bounding box as well.)

Figure 89 illustrates the required computation. Let the distance from the start to a point on the ellipse boundary plus the distance from that point to the goal be D_E. The maximum distance that can be traveled along any path while the cost of that path does not exceed U occurs when the entire path can be traveled at cost C_O. Thus,

$$U = D_E \times C_O$$

$$D_E = U / C_O.$$

Let $d(P_1,P_2)$ denote the Euclidean distance from point P_1 to point P_2. By construction (refer to Figure 89)

$$d(Start,B) + d(B,Goal) = D_E$$

$$d(Start,Goal) + d(Goal.B) + d(B.Goal) = d(Start.B) + d(B.Goal)$$

By definition of an ellipse,

$$d(A,Start) = d(B,Goal)$$

$$\text{Therefore } d(A,Start) + d(Start,Goal) + d(Goal,B) = D_E$$

$$\text{and the ellipse major axis is } D_E = U/C_O$$

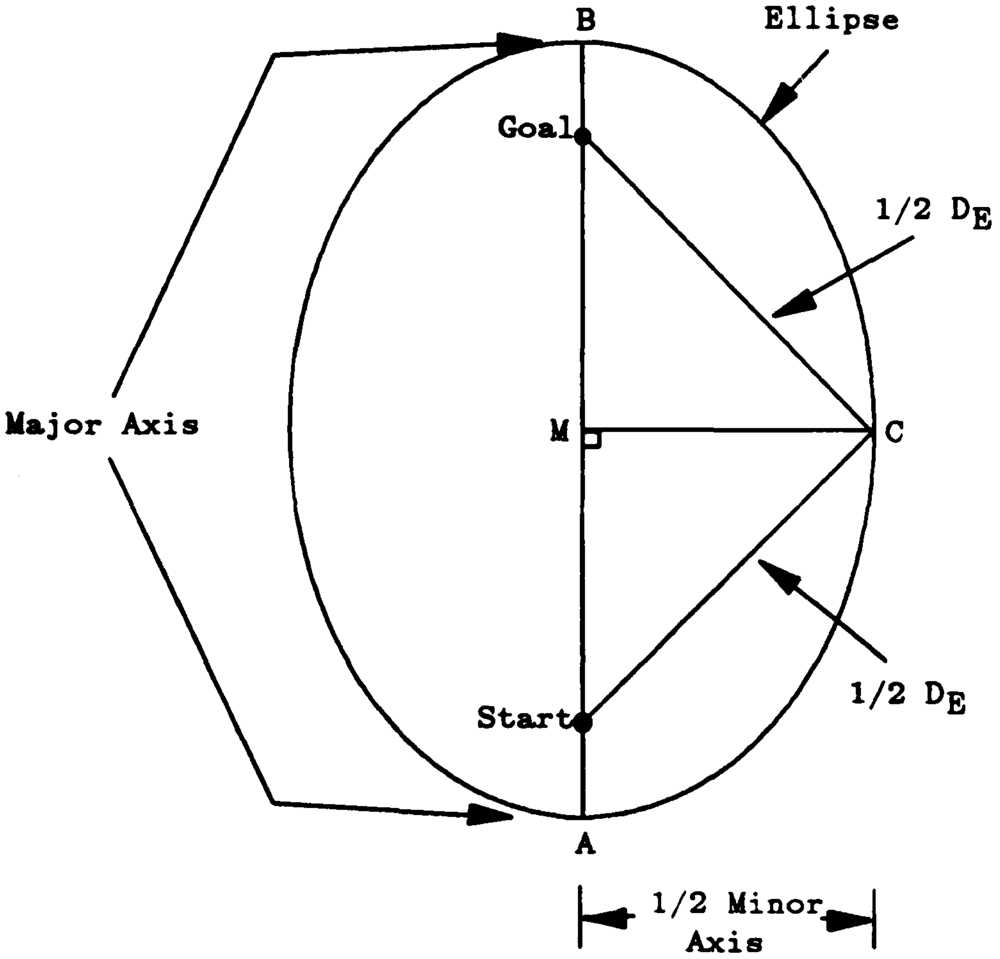


Figure 89. Constructing The Ellipse

Let M be the midpoint of the start-to-goal line segment and let the line segment $M-C$ be perpendicular to the $Start-Goal$ segment. Then, by construction,

$$d(Goal,C) = d(Start,C) = 1/2D_E$$

By the Pythagorean theorem,

$$d(Start,M)^2 + d(M,C)^2 = \left(1/2D_E \right)^2$$

$$d(M,C)^2 = 1/4D_E{}^2 - d(Start,M)^2$$

$$d(M,C) = \left(1/4D_E{}^2 - d(Start,M)^2 \right)^{1/2}$$

Let $D_{SG} = d(Start,Goal)$. By definition, $1/2D_{SG} = d(Start,M)$. Then, substituting,

$$d(M,C) = 1/2\left(D_E^2 - D_{SG}^2 \right)^{1/2}$$

Since the ellipse minor axis is $2d(M,C)$ it is equal to

$$\left(D_E^2 - D_{SG}^2 \right)^{1/2}$$

Thus, we have the formulae to construct either an ellipse or a rectangular bounding box suitable for physically limiting the problem search space. The bounding box is the preferred structure since wedges are defined so that they terminate when they intersect the limit of the search space. Computing the intersection of two line segments is simpler than computing the intersection of a line and an ellipse.

The bounding box defines the physical space over which the search is to be conducted. Accordingly, only those homogeneous-cost regions (including obstacle regions) that lie, at least partially, within the bounding box need be considered during search. Ray tracing is a fundamental operation in the Snell's-law-based algorithm. That is, given a point on a path and a heading from that point, ray tracing is the process of finding the intersections of the path and region boundaries (or an intersection with the bounding box). Once the physical limits of the search space have been defined (by construction of the bounding box), the algorithm must examine that space to construct a set of boundaries (and vertices) that it contains in order to limit the total set of boundaries considered during ray tracing. It is often true that a region boundary lies within the bounding box while one of its endpoints does not. In these cases, the algorithm creates an artificial vertex called a ***boundary point*** where the boundary intersects the bounding box. Then, the boundary points, the region vertices within the bounding box, and the goal point form the set of search point⁻, as defined in Section IV.B. Note that the search point set also defines the boundaries of homogeneous-cost regions and obstacle regions that must be considered during ray tracing. We denote these boundaries as the ***search boundaries.***

The next step required of the algorithm as depicted in Figure 88 is the creation of an initial wedge. At this point in the search process, very little is known about probable locations for the optimal-cost solution path (assuming that stored paths, as described in Section IV.I are not available). Thus, the initial wedge should not eliminate any portions of the bounding box from the search effort. To ensure that no area is overlooked, two initial wedges are created so that they do not overlap but the two of them together contain all the area within the bounding box.

The initial wedges can be created by a simple procedure. Assume that the straight-line start-to-goal path has a heading of H_{SG} (refer to Figure 90). Form

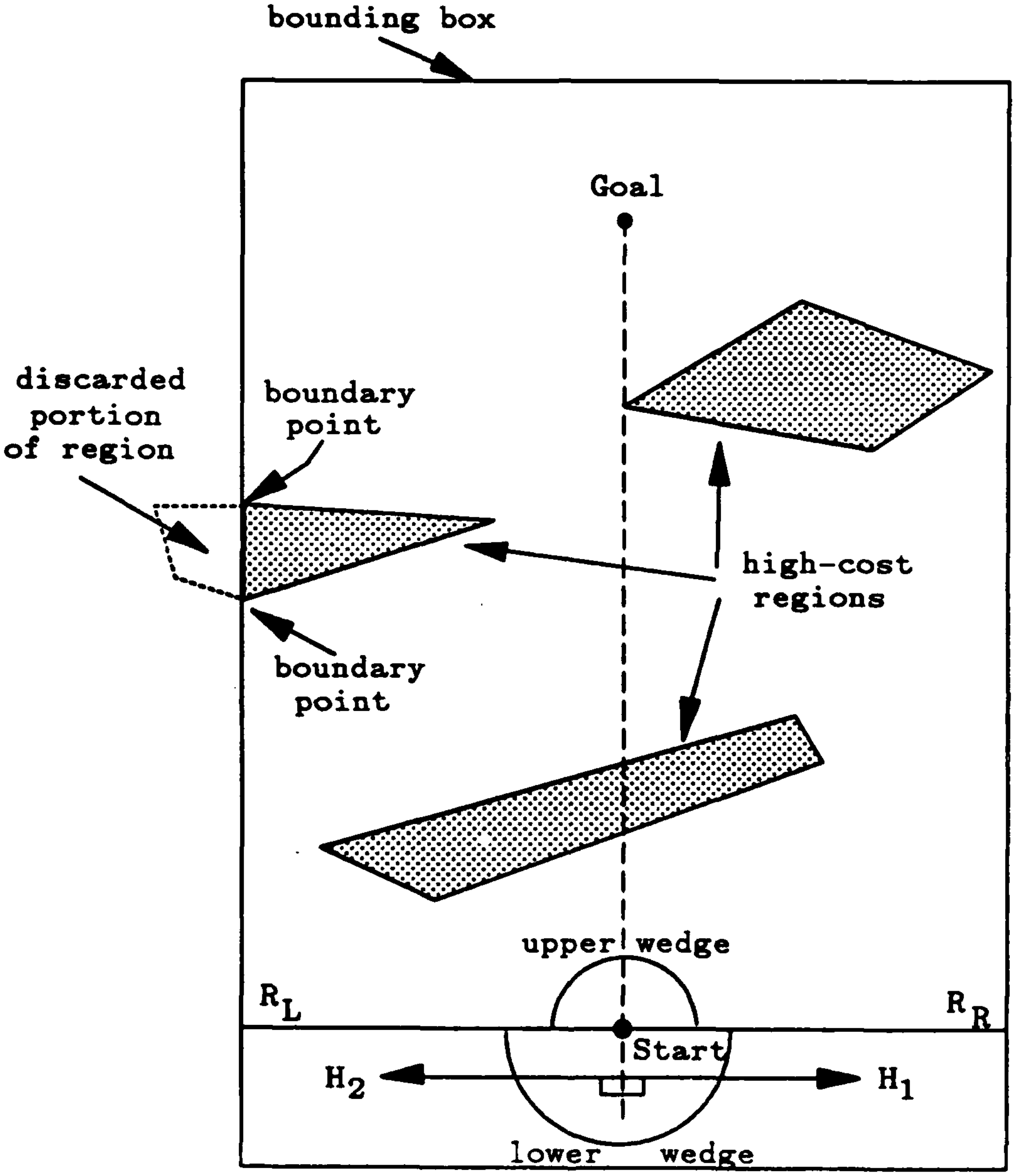


Figure 90. Initial Wedges

heading H_1 by rotating clockwise 90 degrees from H_{SG}. Conduct ray tracing on a path from the start having heading H_1 until it terminates (by, for example, intersecting an obstacle or one side of the bounding box). Call this path R_R. Form heading H_2 by rotating 90 degrees counter-clockwise from H_{SG} and conduct a similar ray tracing, resulting in path R_L. Paths R_L and R_R define two initial wedges, denoted as the upper and lower wedge in Figure 90. The upper wedge has R_L as a left boundary and R_R as a right boundary. The lower wedge has R_R as a left boundary and R_L as a right boundary. Both wedges have the start location as a wedge tip. Clearly, the two wedges cannot overlap and together, they contain every point within the bounding box.

Once the two wedges are created, they can be rated by the criteria established in Chapter IV. Point-to-point lower-bound cost evaluations (from the wedge tip to the goal) can be used for both wedges since it is impossible for R_L and R_R to form a K-explored wedge for K > 0. The ordered agenda is initialized to contain both the upper and lower wedges which have the same lower-bound cost evaluation.

Finding a feasible solution, constructing the bounding box, establishing the search point and search boundary sets, and creating the agenda fulfill the initialization requirements. The general algorithm can now be applied to solve a specific instance of the general-case weighted-region problem.

C. SELECTING A WEDGE FOR REFINEMENT

The agenda is a list of wedges ordered in terms of increasing lower-bound cost evaluations for possible start-to-goal paths that they might contain. Given such an ordering scheme, selecting the most favorable wedge for refinement amounts to choosing the first wedge on the agenda. The chosen wedge must be removed from the agenda, in the spirit of the A* algorithm. Recall that A* also requires the maintenance of a Closed list (a set of previously expanded nodes) to prevent duplication of search effort and infinite cycling. Cycle avoidance is also important to this version of A* search although there is a more efficient method to achieve it than through use of a Closed list. As explained below (in Section V.G) cycles are prevented by exercising care when new wedges are added to the agenda.

Our algorithm utilizes a termination criterion slightly different from that generally used in A* search. The A* search strategy normally terminates when the first element on the agenda (i.e., the most favorable partial solution) is a complete solution. Our algorithm does not store partial solution paths on the agenda; rather, partially explored physical spaces (in the form of K-explored wedges) are retained. However, our algorithm does maintain an upper-bound evaluation that corresponds to the cost of the best known start-to-goal path found so far. The termination criteria is met when the first element on the agenda (i.e., the wedge with the lowest lower-bound cost evaluation) has a cost evaluation that exceeds the upper-bound cost evaluation. The agenda is ordered, so every other element on the agenda must also have an evaluation greater than the upper bound on the cost of the optimal solution path and the search can stop since it can no longer be profitable. Search can also halt when the agenda is empty. In either case, the known solution that has least cost is returned as the optimal solution. A feasible solution is always known since one must be computed during initialization. Optimality of the solution is guaranteed by the correctness of the pruning criteria developed in the preceding chapter.

D. LOCATING THE CLOSEST UNSOLVED SEARCH POINT

Recall that refining a wedge is accomplished by determining the closest unsolved search point within the wedge, calculating the Snell's-law path from the wedge tip to that point, splitting the path into two paths at the search point, and selecting pairs of paths to define new sub-wedges. Thus, locating the closest unsolved search point is fundamental. The procedure to find this point can be aided by descriptions of the paths that define the wedge boundaries.

Our algorithm describes individual paths as a series of turn points. This is intuitively appealing since the path segments must be (straight) line segments within each homogeneous-cost region. Heading changes only occur at boundary-crossing episodes (when the path intersects a region boundary) or when the path includes a diffraction vertex (in this sense, diffraction vertices correspond to null boundary-crossing episodes). For each boundary-crossing episode, the X and Y coordinates of the path intersection with the boundary, a designation of the boundary being crossed, and the cost rate along the path just before intersecting the boundary are recorded. At diffraction vertices, the coordinates of the vertex, a special designation of the null boundary, and the cost rate just before reaching the vertex are recorded. Including the cost information is helpful for several reasons. It allows path costs to be easily computed, referencing only the path itself. Also, note that two cost rates must be associated with each boundary, one inside a region and one outside the region. Knowing one cost rate allows the other to be easily determined.

This representation for the wedge boundary-defining paths facilitates locating the closest unsolved search point within the wedge. We accomplish this by forming polygons that describe portions of the wedge and inspecting those polygons to find any search points that they contain. Procedurally, the process starts at the wedge tip and follows along the wedge-defining paths up to their first non-null boundary-crossing episode. If the two paths intersect the same boundary at their first boundary-crossing episode (their path descriptions designate the same boundary other than the null boundary), we form a triangle whose vertices are the wedge tip, the first intersection on the left path, and the first intersection on the right path (as illustrated by the striped region in Figure 91). Then, those search points within the triangle can be determined as explained below. (Note that any search points on either wedge-defining path should not be considered. Otherwise, duplicate paths will result.) If such points exist, the one closest to the wedge tip is selected as the closest unsolved search point.

If the triangle contains no search points, a second polygon, again based on the two wedge-defining paths, can be formed. Suppose that the two paths intersect the same boundaries at their first and second boundary-crossing episodes, as in Figure 92. Then a quadrilateral, whose vertices are the first two intersections of the left path and the first two intersections of the right path, can be formed. Again, this polygon can be examined to determine those search points that it contains. Of those points, the single point closest to the first boundary (i.e., the

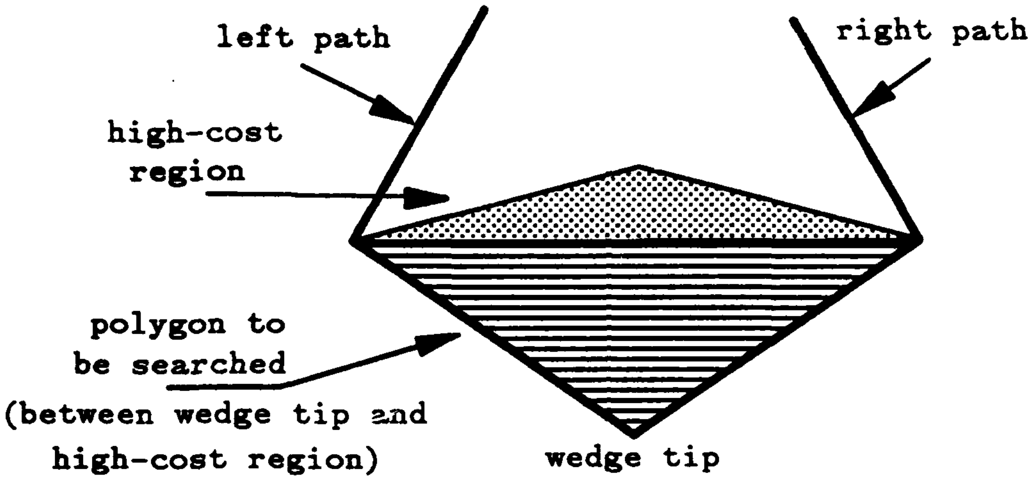

left path
right path
high-cost region
polygon to be searched (between wedge tip and high-cost region)
wedge tip

Figure 91. Triangular Search Polygon

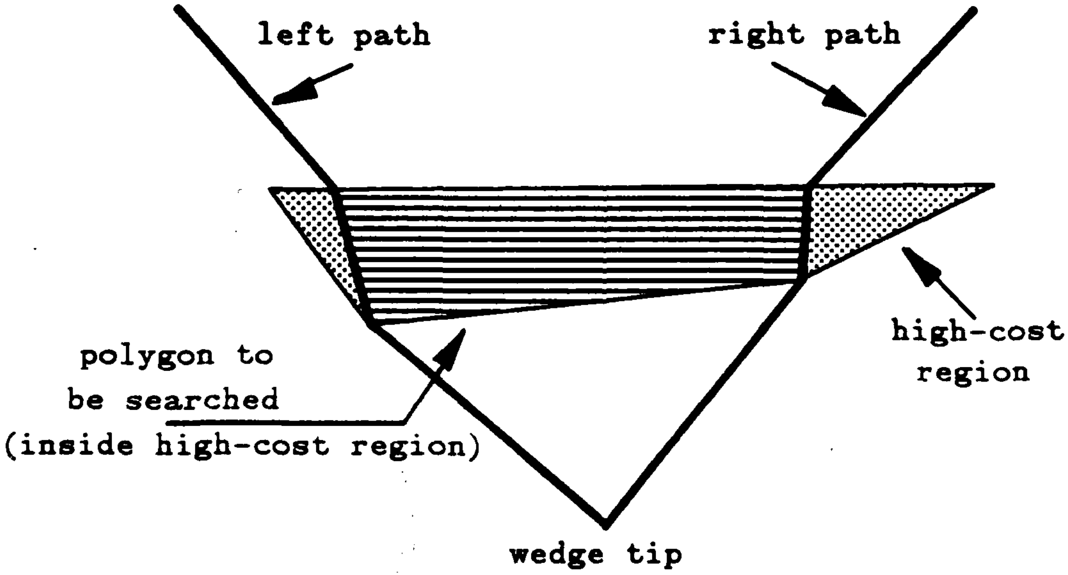
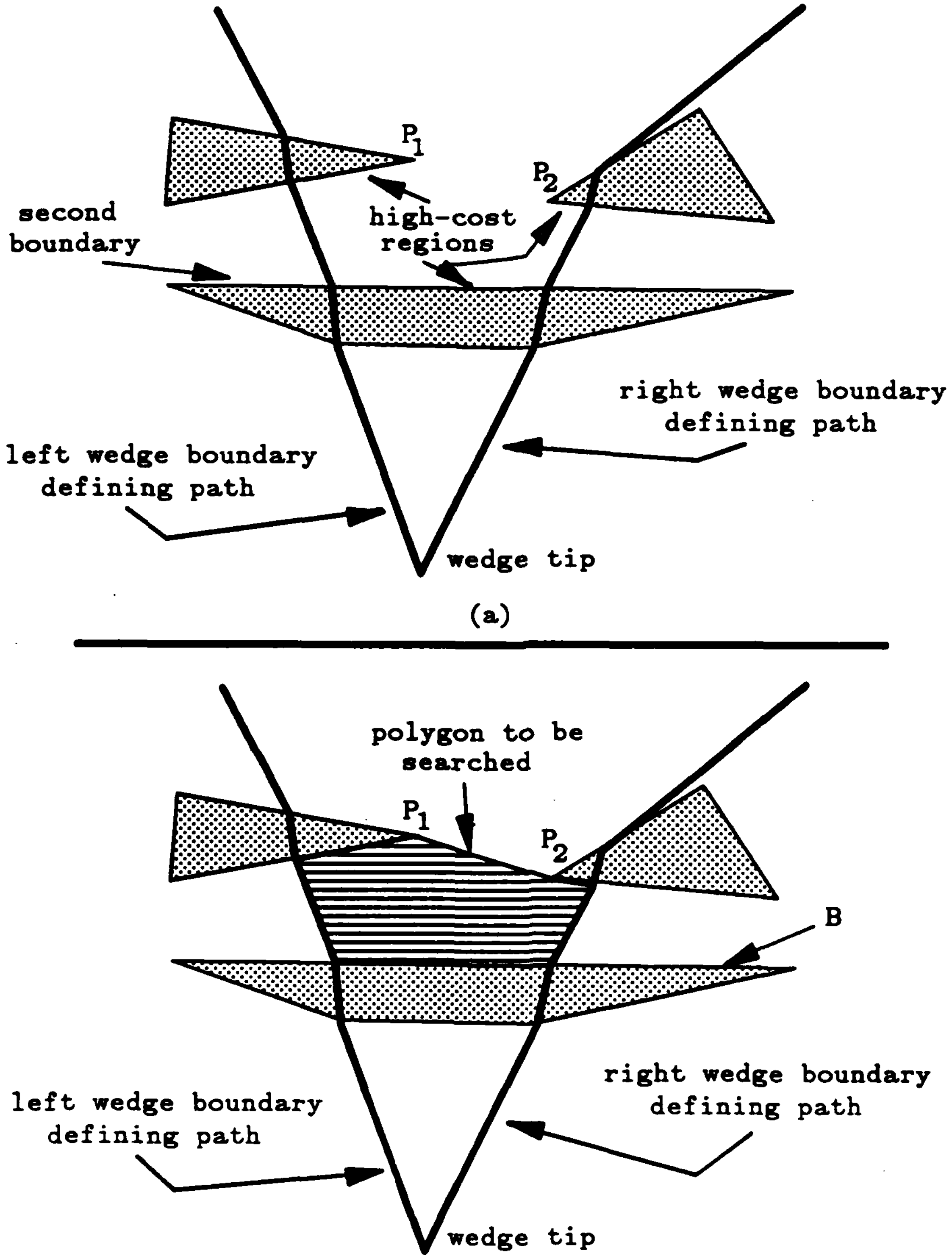


Figure 92. Quadrilateral Search Polygon

boundary that causes the first boundary-crossing episode for both wedge-defining paths) can be selected as the closest unsolved search point. If the polygon contains no search points, a new quadrilateral based on the second and third boundary-crossing episodes of the wedge-defining paths can be constructed. Here, proximity to the second boundary (instead of the first) is used to determine the closest search point. This process can continue until the wedge-defining paths intersect different boundaries or terminate. If no closest unsolved search point has been found, the wedge is empty and need not be searched.

In most cases, the wedge-defining paths will intersect different boundaries at some point. When this occurs, one endpoint of each of the two different boundaries must be inside the wedge and must be unsolved for that wedge. (Note that the different boundaries intersected may share a common endpoint.) Here, either a pentagon (if the wedge tip is a vertex of the polygon that will be inspected to determine the search points that it contains) or a hexagon (otherwise) can be formed. This polygon must contain an unsolved search point. Figure 93 illustrates the hexagonal case. Figure 93(a) depicts the two paths as they intersect different region boundaries at their third boundary-crossing episode. The striped hexagon in Figure 93(b) is the polygon used to determine the closest unsolved search point. Here, vertex P_2 is the closest search point to boundary B and is thus the closest unsolved search point within the wedge. (Proximity to the wedge tip can be misleading when the wedge tip is not in the polygon.)

In the process of finding the closest unsolved search point, the wedge may be found to be empty (i.e., contain no search points), making further refinement impossible. Clearly, such wedges can be pruned from the search. Other


P_1
P_2
second boundary
high-cost regions
right wedge boundary defining path
left wedge boundary defining path
wedge tip
polygon to be searched
P_1
P_2
B
right wedge boundary defining path
left wedge boundary defining path
wedge tip

(a)

(b)

Figure 93. Hexagonal Search Polygon

opportunities for pruning also exist. Suppose both wedge-defining paths intersect the same obstacle, on any of the obstacle's boundaries. If the wedge contains no search points between the wedge tip and the obstacle, it can be pruned since, by Theorem 2, every other path within the wedge must also intersect the obstacle. Pruning according to Lemma 11 is also easily accomplished by the procedure designed to find the closest unsolved search point. Proceeding one boundary at a time also facilitates determining if and when wedge-defining paths intersect. Recall that such intersections terminate wedges.

The efficiency of the point locating process can be improved. It is not necessary to examine every possible polygon that can be formed by consecutive boundary-crossing episodes within the wedge. Some of these polygons close to the wedge tip, in the general case, will have already been examined. A wedge is refined from a parent wedge. Thus, some polygons that can be formed within a wedge are simply smaller versions of those that have already been examined within a parent wedge. If a polygon within the parent wedge has already been examined and found to contain no search points, there is no point in re-examining a (smaller) subdivision of it at some later time. Thus, marking portions of wedges as already examined (based on the examination of parent wedges) can save some effort later. (This marking could be accomplished by storing another data item in the path descriptions of the wedge-defining Snell's-law paths.)

Indexing the members of the search point set by their position on the area-cost map can enhance efficiency. Such indexing allows some search points to be quickly eliminated from the search for the closest unsolved search point. As an example, a polygon located entirely in the lower left corner of the map cannot contain search points from the upper right corner. Intelligent indexing can make such occurrences readily apparent. As an example, we can divide the area-cost map into disjoint blocks and store the search points as sets of points inside specific blocks. That is, all of the search points located inside a single block can be stored together. The search point set could be organized in this manner by using two *hash functions*, [Ref. 46] one for X coordinates and one for Y coordinates. Then, we can determine the blocks touched by a wedge (or portion of a wedge) by hashing on the minimum and maximum X and Y coordinates on the Snell's-law paths defining the wedge. Retrieving only those blocks containing search points in the vicinity of the wedge could decrease the total number of search points considered during the search for the closest unsolved search point. Also note that region vertices that are already on either wedge-defining Snell's-law path are not considered in selecting the closest unsolved search point. Selecting any of these points would result in self-intersecting wedges that could not lead to optimal-cost solution paths.

Once a polygon has been formed, there must be some relatively efficient method to determine those points from a given set, if any, that are inside that polygon. (For general-purpose methods of positioning a point with respect to a polygon, see [Ref. 47, p.330].) The polygons may be non-convex (as in Figure 93), somewhat complicating the issue. Our algorithm relies on a simple strategy to compute interior points. For each candidate point, we construct a horizontal line segment from that point to one side of the bounding box. If this segment has an odd number of intersections with the constructed polygon, the point is inside the polygon. If the number of intersections is even, the point is outside. A problem arises in this scheme if the candidate point has the same Y coordinate as some vertex of the polygon [Ref. 48]. When the X coordinate of the vertex is on the horizontal line segment, the other endpoints of the polygon sides that share the conflicting vertex as a common endpoint must be examined. If both of these endpoints are either above or below the horizontal line, no intersection is tallied for those two polygon boundaries. Otherwise, one intersection is counted and neither of the two boundaries is considered further for that candidate point. (This assumes that neither of the two endpoints also has the same Y coordinate as the candidate point. If this is not true, the procedure must "follow around" the polygon sides until some vertex has a different Y coordinate.)

The rationale for choosing the *closest* unsolved search point is based on the requirement to find Snell's-law paths. By the above procedure to find these points, the wedge must be K-explored up to the closest unsolved search point. By Theorem 2, the Snell's-law path from the wedge tip to that point must intersect the same K boundaries in the same order. Thus, ray tracing is facilitated by knowing in advance the exact sequence of boundary-crossing episodes that must occur. This is valuable knowledge because it allows the ray-tracing routine to know, without search, the next region boundary to be intersected after each boundary-crossing episode on the Snell's-law path.

There is no known closed-form solution yielding the Snell's-law path between two arbitrary points. Instead, iterative ray tracing must be performed until a path is constructed that comes acceptably close to the desired search point. Developing a suitable iterative search strategy is the topic of the next section.

E. ITERATIVE SOLUTION OF SNELL'S LAW

A solution to the problem of finding a Snell's-law path between two points is a path that begins at the first point and travels within some specified tolerance of the second point, obeying Snell's law at each boundary-crossing episode along the way. In this sense, an iterative solution to a Snell's-law point-to-point problem is a minimization problem, minimizing the distance from a specific path to a known point. Before the iterative search begins, we must know that the solution path is within a specific wedge. The wedge is defined by left and right boundary Snell's-law paths. Thus, the wedge-defining paths *bracket* the Snell's-law solution path. That is, one Snell's-law path lies entirely to the left of the solution path and the other Snell's-law path lies entirely to the right of the solution path. By Theorems 1 and 2, we know that there is a single Snell's-law path, within the wedge, that exactly contains the search point.

For notation, we define $newdist(X)$ as a function that returns the minimum distance from a Snell's-law path to a (constant) point, given X, a description of the first boundary-crossing episode on the path. Figure 94 illustrates the definition. Here, the region boundary is indicated by the heavy line and P is the intersection point of the first boundary-crossing episode for Snell's-law path R_P. Let R_L and R_R be the Snell's-law paths defining the wedge and let G be the closest unsolved search point within that wedge. Let d_1 be the distance from P to P_L, R_L's first boundary-crossing episode. Let d_2 be the distance from G to P_{RP}, the intersection of R_P and an imaginary line that is parallel to boundary B and that goes through point G (the dashed line in the figure). Then, by definition,

$$newdist(d_1) = d_2$$

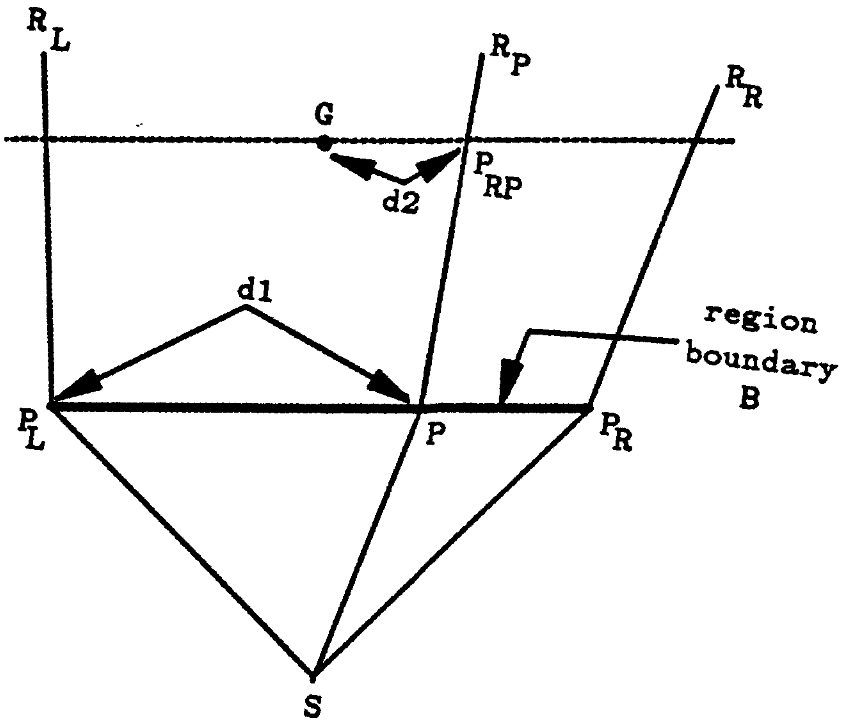


Figure 94. Newdist(d1) = d2

(We note that if there are no region boundaries between S and G, then *newdist*(X) is undefined. In this case, however, there is also no requirement for search. In general, when two points are within the same region, the best path between them is a straight line. Exceptions to this rule can occur when critical-angle reflection paths are involved. Such paths are discussed in Section V.H below.)

The function *newdist*(X) is very similar to the *dist*(P) function defined in the statement of Lemma 5. The only difference between the two functions is in the boundary selected to measure P and X. This difference is trivial since a P distance measured along the first boundary intersected by a Snell's law path uniquely determines the X distance for that path, measured on the Kth boundary. Thus, we know that *newdist*(X) is a quasi-convex function having a single minimum value. Based on Theorem 2, this minimum occurs when *newdist*$(X) = 0$ and X determines a Snell's-law S-to-G path. Thus, any standard one-dimensional search technique, such as bisection or golden section search, is suitable for solving a Snell's-law point-to-point path problem. These standard line search techniques are sufficiently powerful to find minimum values for arbitrary continuous functions. We prefer a method that exploits knowledge about the *newdist*(X) function to quickly converge to a minimum.

Figures 95 through 99 illustrate a new function, *close*(X), which is a modified version of *newdist*(X). This new function is a modification of *newdist*(X) because the value of d_2 is taken to be negative for those paths traveling to the left of the goal and positive otherwise. In this case, the single minimum value of *newdist*(X) occurs at the point where *close*(X) intersects the horizontal axis. We have defined *close*(X) in this manner for illustrative purposes. In each figure, d_2 is plotted along the vertical axis while the horizontal axis reflects d_1 values. The Snell's-law paths and the cost regions used to generate the curves are shown in the inset for each figure. In the inset, paths enter the (bottom) high-cost region at uniform intervals along its lower boundary. The goal used in the computation is not shown. Note that the only effect of the goal location (within same the wedge) is to shift the horizontal axis up or down.

In Figure 95, the two boundaries of the high-cost region are parallel. In this case, *close*(X) is linear. Figures 96 and 97 show that when Snell's-law paths intersect non-parallel high-cost region boundaries, a "bend" can result in the

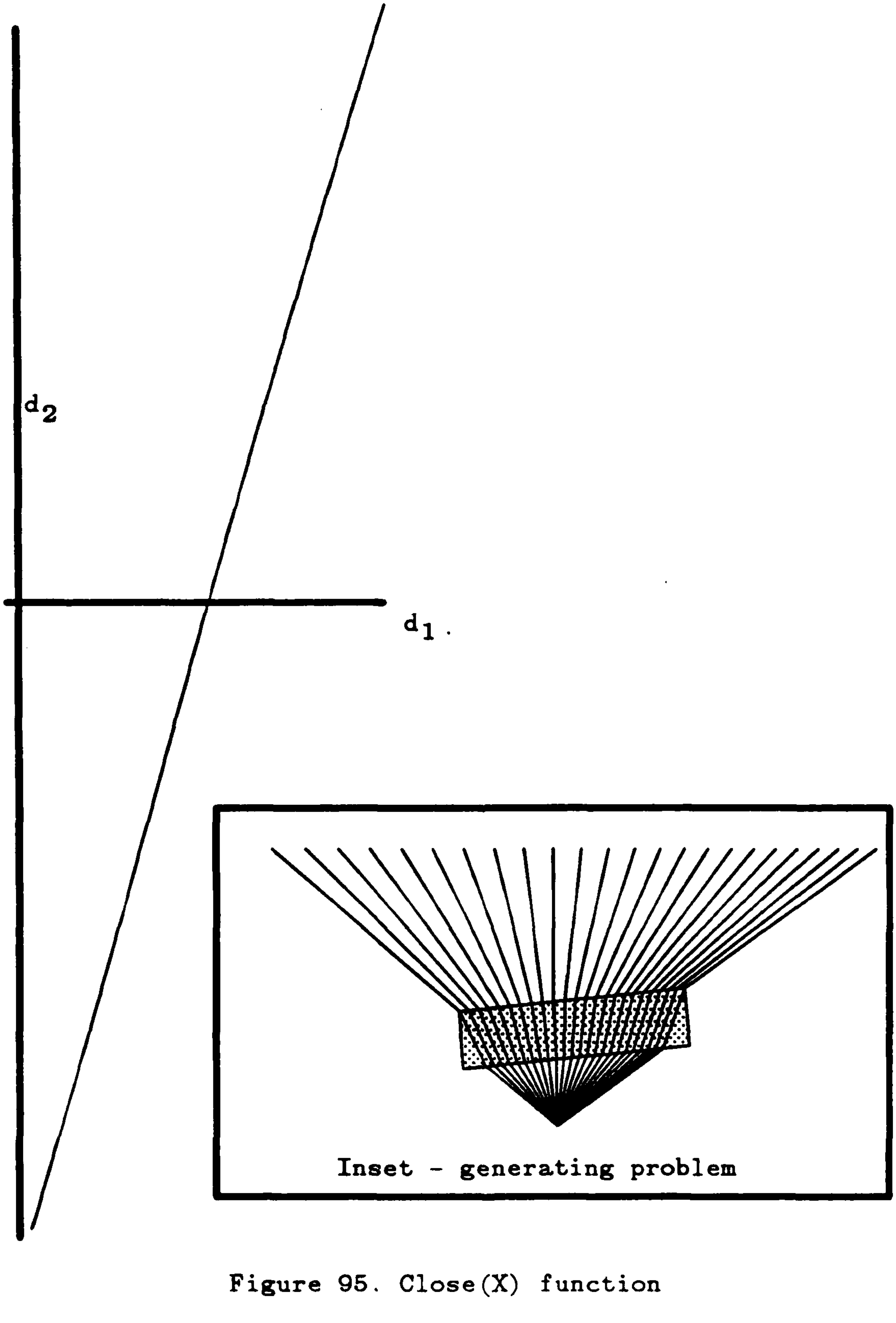


Figure 95. Close(X) function

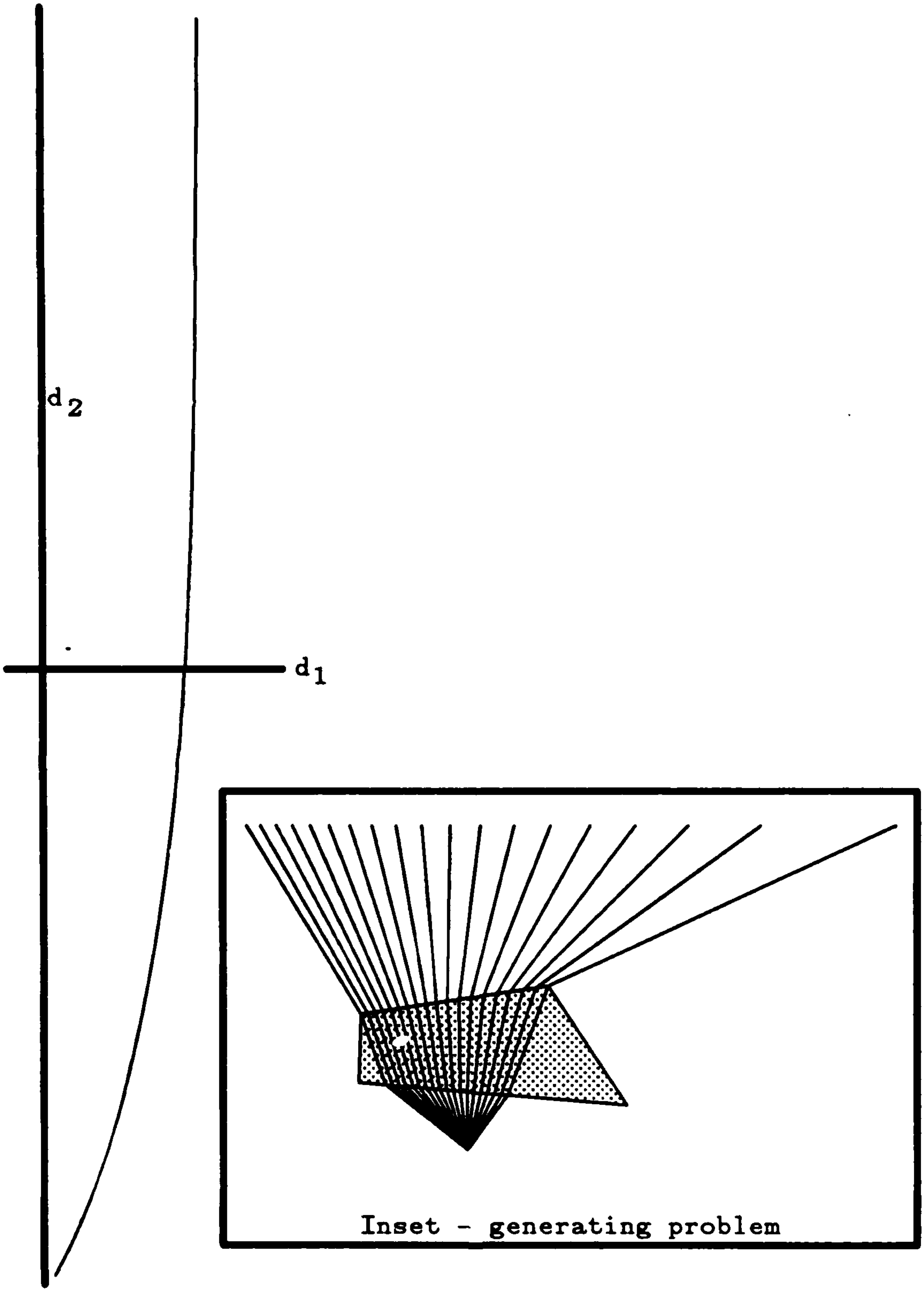


Figure 96. Close(X) Function

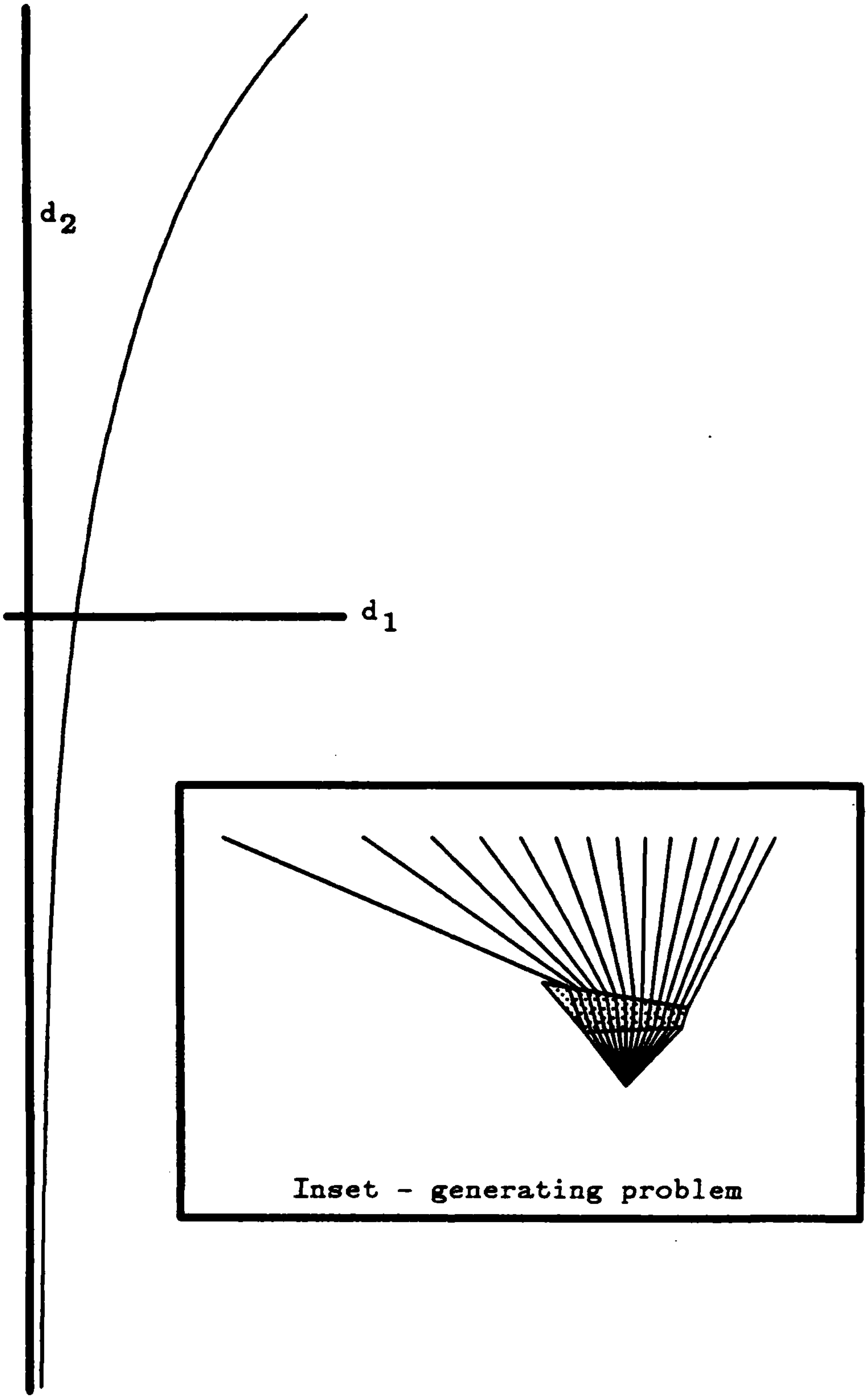


Figure 97. Close(X) Function

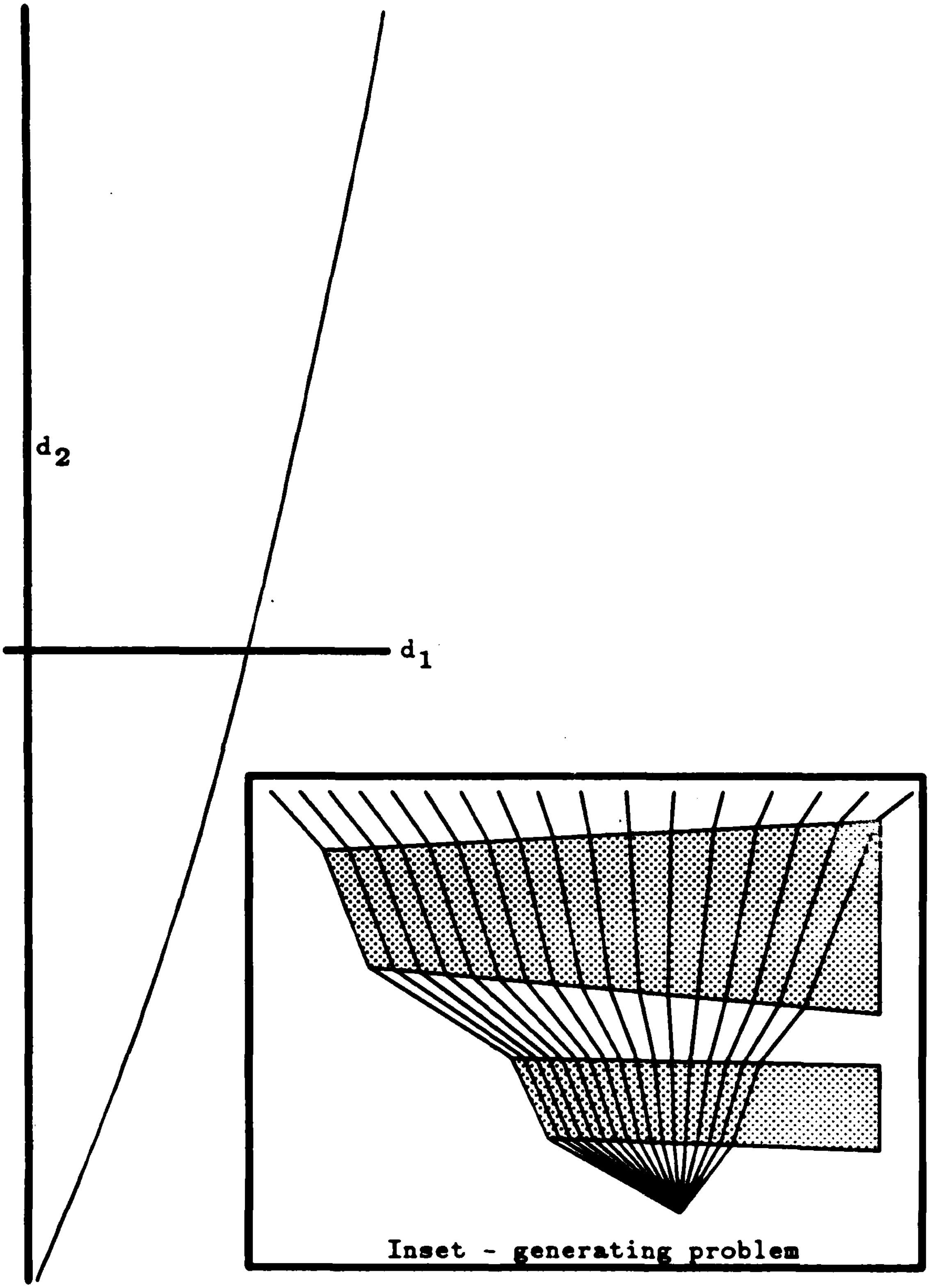


Figure 98. Close(X) Function

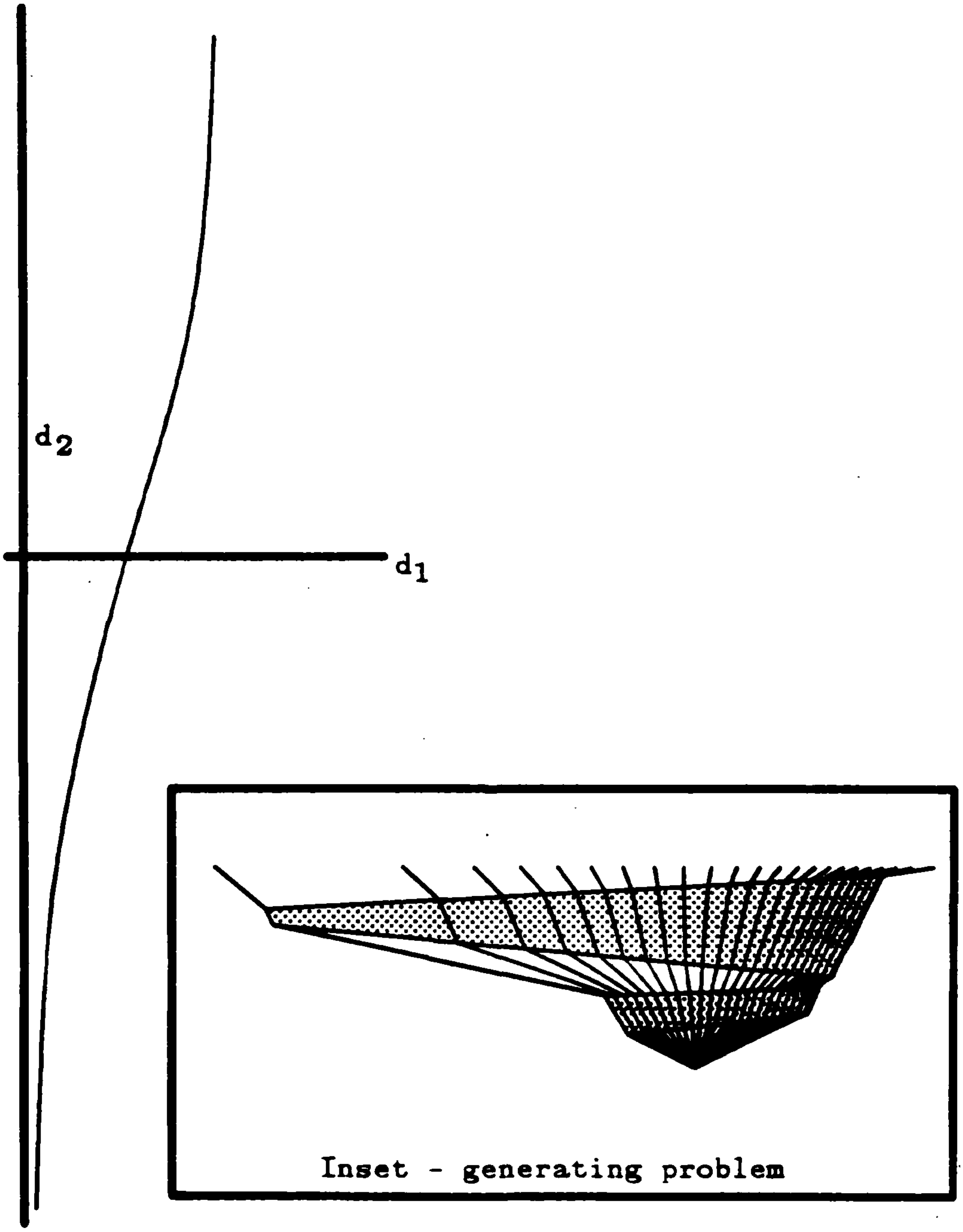


Figure 99. Close(X) Function

close(X) curve. The amount of the bend is determined by the range of rotation amounts of the Snell's-law paths as they exit the last boundary. The rotation amount is invariant in the number of boundary-crossing episodes and is limited to a total range of 180 degrees. Figure 98 was generated by paths crossing two high-cost regions. Note that the *close*(X) curve here is similar to the preceding curves. However, multiple regions can have the effect of introducing multiple bends into the curve, as illustrated by the problem shown in Figure 99. In each figure, large portions of the curve are nearly linear which suggests that a secant search, as illustrated in Figure 100 is an suitable iterative search strategy.

An important variant of secant search is known as the false position method [Ref. 49]. The technique relies on two bracketing values to interpolate a value to be used on the next iteration. The interpolated value should result in a path that lies between the two bracketing paths. Thus, on each iteration, either the left or right bracket value can be replaced, resulting in a bracket interval that becomes progressively smaller until an optimal solution (within some specified tolerance) is found. False position searches of this nature are guaranteed to converge.

Convergence of false position searches can be slow when the minimum is constantly approached from only one side. The curve in Figure 100 is the type that yields this kind of slow convergence. Here, the right side bracket value is replaced on every iteration and the left side bracket value remains unaltered for the entire search. An iterative search technique can converge more quickly when alternate bracket values are replaced on successive iterations. We facilitate this behavior by adding some heuristics to the basic false position method.

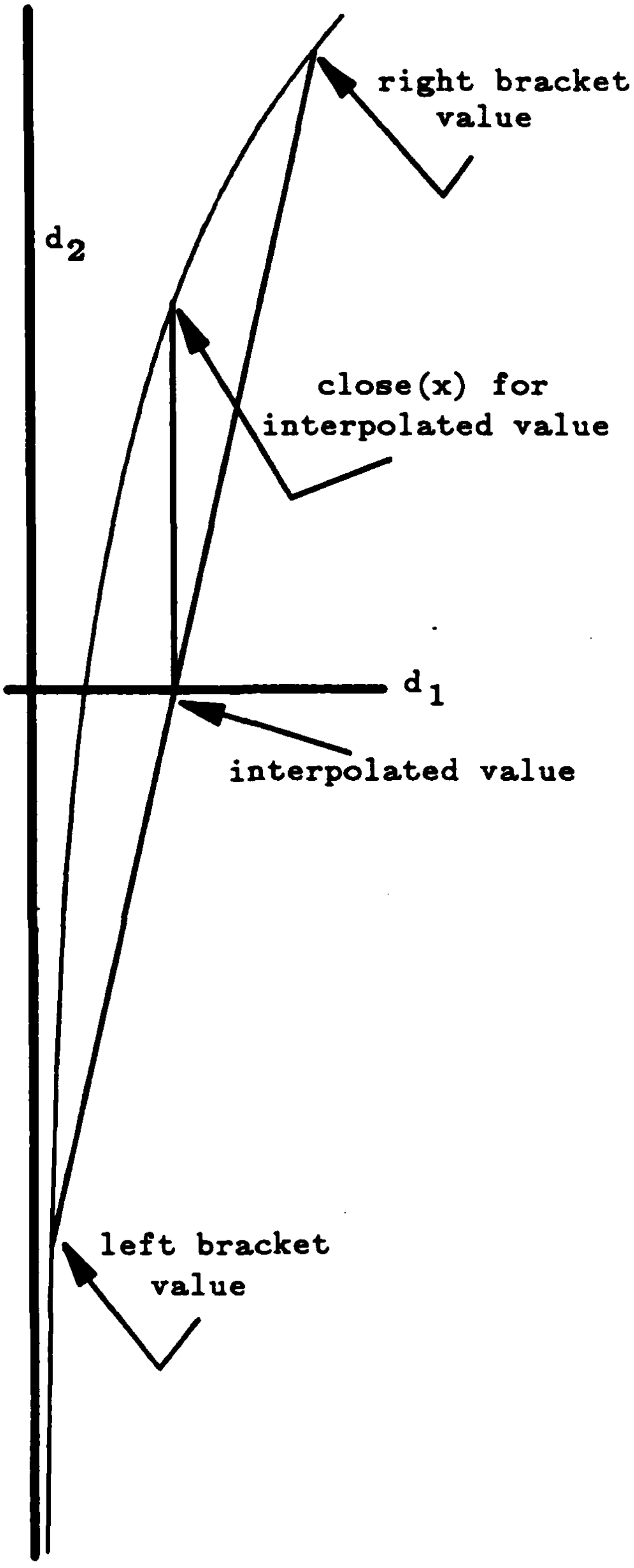


Figure 100. A False Position Method, Secant Search

The line search technique that we use is a modification of false position search that keeps track of which bracket value is replaced on each iteration. When the same value is to be replaced on successive iterations, the algorithm "boosts" the interpolated value, based on the amount of error in the last interpolation. (See Figure 101.) When this fails to alter the replaced bracket value after three iterations (simply a heuristic), the algorithm returns a bisection between the unreplaced bracket value and the interpolated value, as in Figure 102. Generally, the bisection causes the alternate bracket value to be replaced. We denote this method of search as ***heuristic false position*** search.

Table 13 compares the performance of bisection, golden section, standard false position method and the heuristic false position method searches. Each technique was used to solve the same set of 100 point-to-point Snell's-law problems. These problems were chosen at random, the only requirement being that the path between the selected points must intersect all high-cost region boundaries in the problem. When the $close(X)$ function returned a value of 0.01 units or less, the problem was considered to be solved. The mean, standard deviation, minimum and maximum columns of Table 13 refer to the number of iterations required to solve each problem (note that a single high and low number of iterations for each method was discarded). The test problems featured different cost ratios between homogeneous-cost regions, region configurations, and number of boundary-crossing episodes on the path. Different initial bracket values were also used. The standard false position method was most affected by altering initial bracket values, due to its "one sided" approach to the minimum. The performance of the

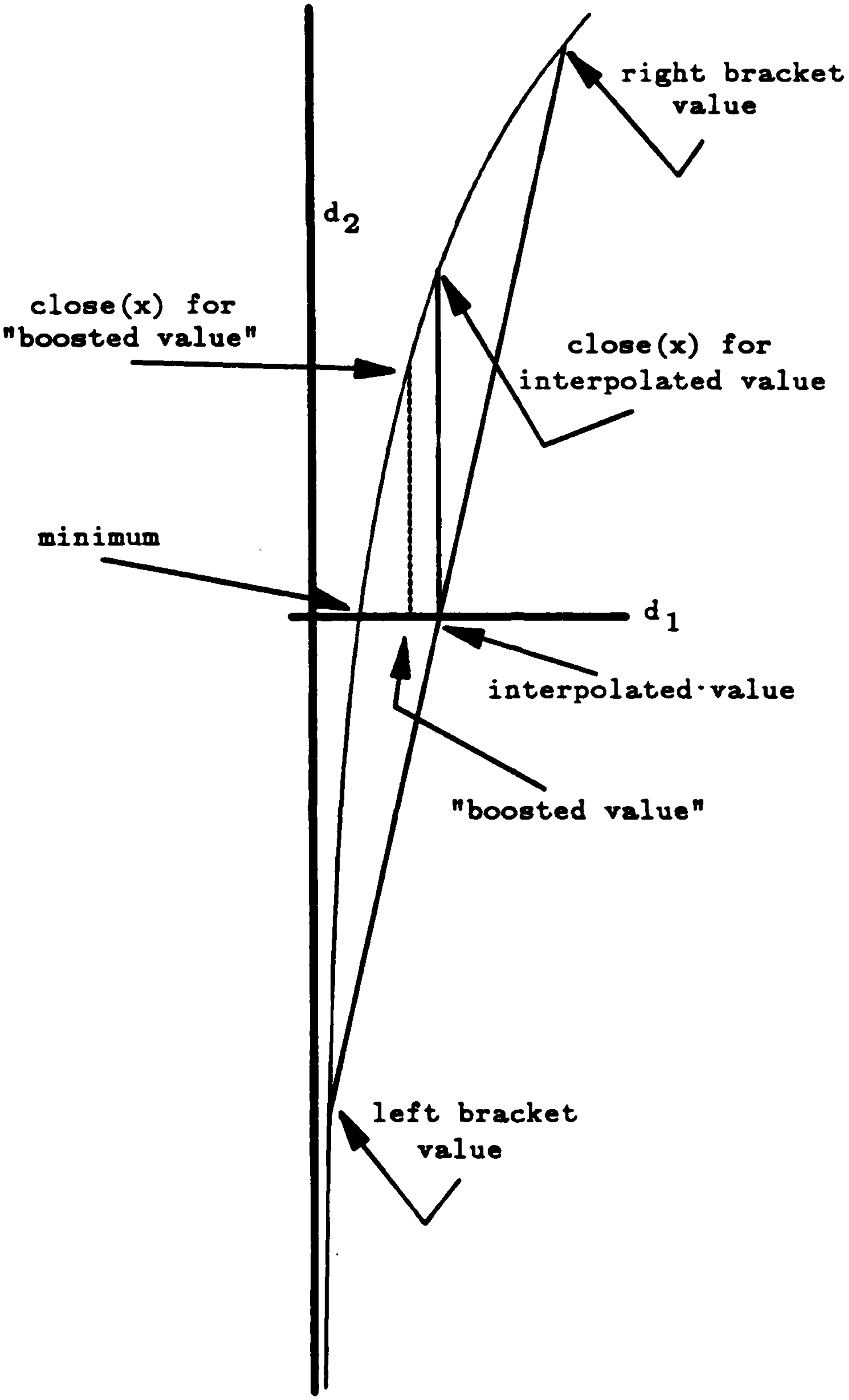


Figure 101. Boosting Interpolated Values

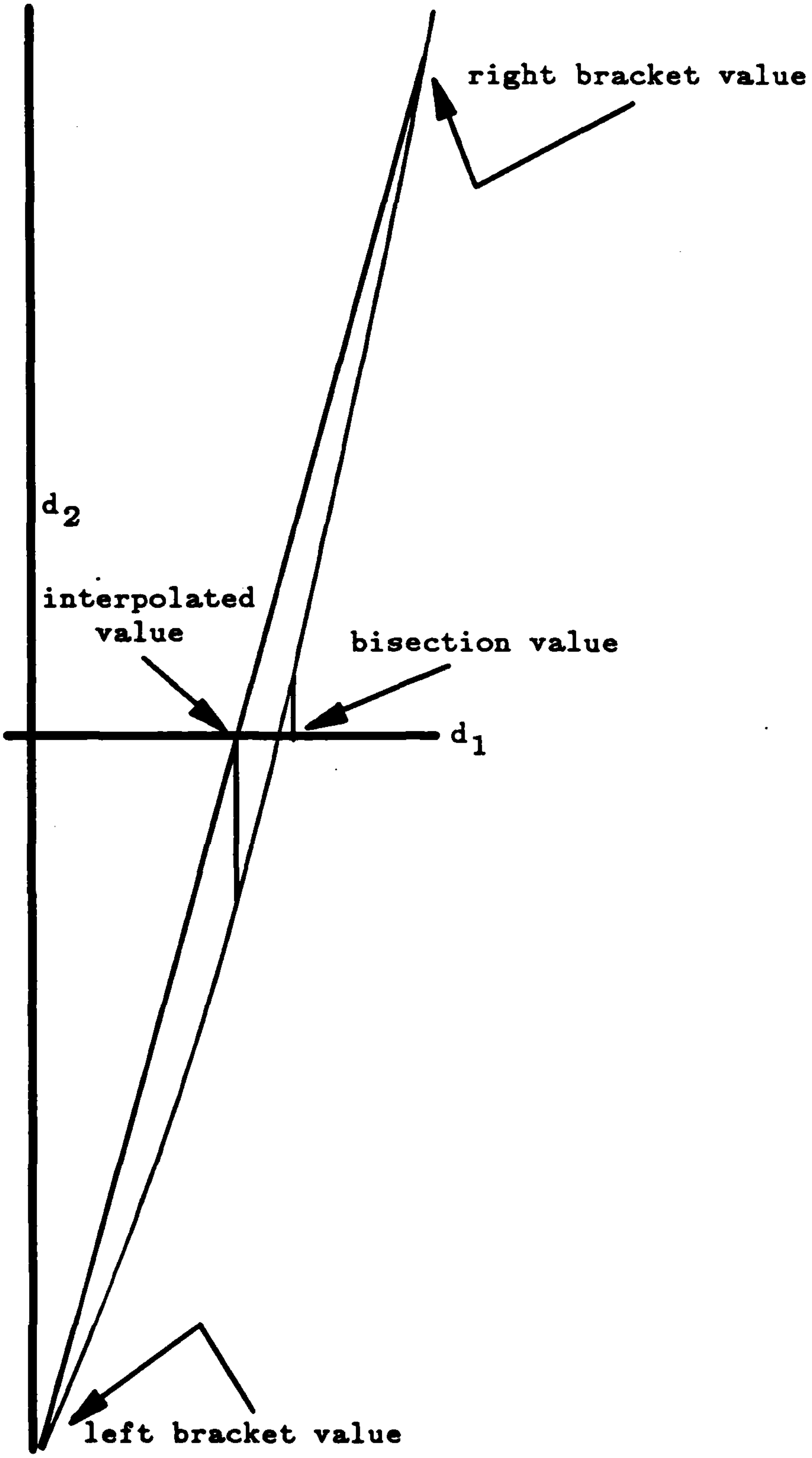


Figure 102. Bisection And Interpolation

heuristic false position method supports its selection as the line search technique used in our algorithm.

TABLE 13
ITERATIONS REQUIRED TO CONVERGE WITHIN 0.01 UNITS
IN 100 TEST CASES

Search Type	Mean	Standard Deviation	Minimum	Maximum
Bisection	12.48	3.11	6	22
Golden Section	13.14	3.21	6	21
False Position	14.60	16.11	2	76
Heuristic False Position	5.55	2.21	2	11

F. WEDGE REFINEMENT

Once a Snell's-law path to the closest unsolved search point is found, that path can be used to refine the wedge that contains it. Let R_P be such a path to point P. Refinement (in general) is accomplished by "splitting" R_P into two distinct paths at P. There are three possible classifications for P; it can be either a homogeneous-cost region vertex, an obstacle region vertex, or the global goal. The classification of P affects the split operation.

When P is a region vertex (either obstacle or high-cost region), R_P acts as a physical limit for paths to its left and right. Let R_L be a path that lies just to the left of R_P and let R_R be a path that lies just to the right of R_P, all three paths within the same wedge, W. Since P was the closest unsolved search point within W, R_L, R_P and R_R must all intersect the same sequence of K region boundaries up to P. However, since P is a common endpoint of two region boundaries, paths

R_L and R_R must intersect different region boundaries at their K+1'st boundary-crossing episode.

Let P be a high-cost region vertex and let B_1 and B_2 be the two region boundaries that share P as a common endpoint. Assume that R_L and R_R travel infinitesimally close to P (and thus infinitesimally close to R_P as well) on its left and right respectively. Each of R_L and R_R can either intersect both B_1 and B_2, one of them, or neither of them. (We illustrate such intersections in Figures 103, 104, and 105.) Let P_1 be the other endpoint of B_1 and P_2 the other endpoint of B_2. Let R_E be the extension of the last path segment of R_P from P outward. If both P_1 and P_2 lie to the left of R_E then R_L must intersect at least one of B_1 and B_2 and path R_R intersects neither of them. If both P_1 and P_2 lie to the right of R_E, then the converse is true. When P_1 lies to the left of R_E and P_2 to its right, then R_L intersects B_1 (at least) and R_R intersects side B_2 (at least). Figures 103, 104 and 105 illustrate the three cases respectively. Note that in Figure 103 R_E and R_R are collinear and in Figure 104, R_L is collinear with R_E.

At first thought, it seems that there could be three other cases where the region intersected by R_E is a low-cost region instead of a high-cost region. However, note that each region boundary in Figures 103 through 105 can also be thought of as a low-cost region boundary since each boundary separates a high- and low-cost region. Thus, the illustrations are sufficient to cover the general cases. Exceptions to the general cases can occur, as described below.

Figures 103 through 105 depict the general cases of paths R_L and R_R continued infinitesimally close to P. It is possible that both P_1 and P_2 lie on the same

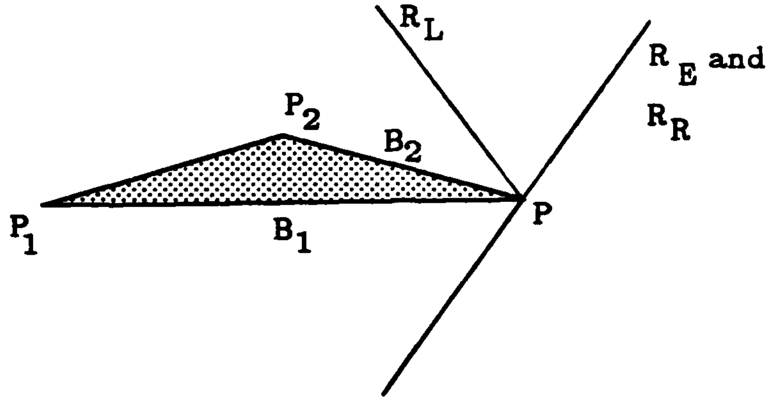


Figure 103. R_L Intersects Both Boundaries

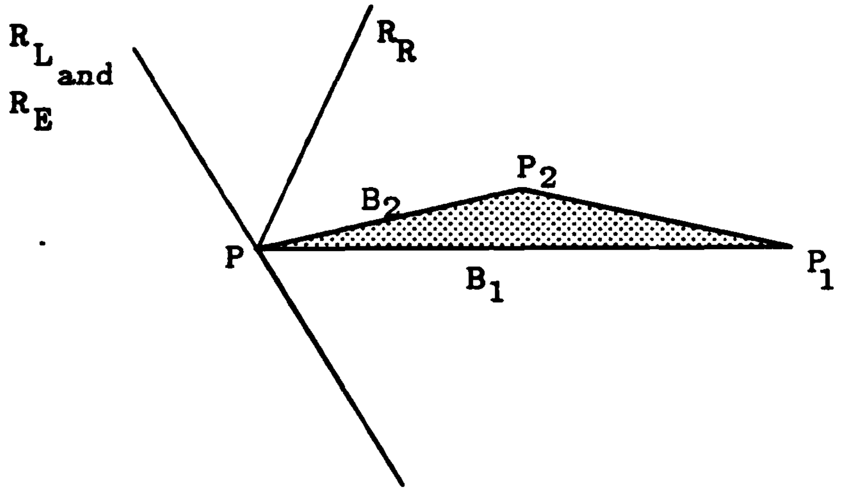


Figure 104. R_R Intersects Both Boundaries

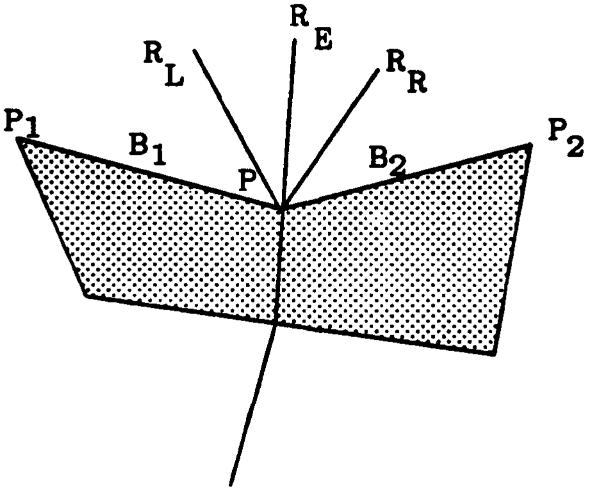


Figure 105. Each Path Intersects One Boundary

side of R_E but that the path also on that side of R_E intersects only one of B_1 and B_2. This situation is illustrated in Figure 106. Similarly, P_1 and P_2 can lie on opposite sides of R_E and a single path can intersect both B_1 and B_2, as shown in Figure 107. To detect these situations, the angle formed about P by sides B_1 and B_2 must be compared to the heading of the new path as it exits the first homogeneous-cost region boundary intersected.

Given that R_L and R_R are correctly traced through the region boundaries sharing P as a common endpoint, then the original wedge W containing R_P can be refined into three new sub-wedges, as illustrated in Figure 108. Suppose that the Snell's-law path defining the left side of W is R_{BL} and W's right boundary is defined by path R_{BR}. Then R_L and R_{BL} form one sub-wedge and R_R and R_{BR} form another. Both of these sub-wedges have the same wedge tip, the wedge tip of W. Also, R_L and R_R together form a new sub-wedge that has P as a wedge tip and R_P as an approach path. Denote these three sub-wedges as W_L, W_R and W_M respectively. In many cases, the middle sub-wedge W_M is empty because R_L and R_R intersect each other at P. Whenever P is a vertex at a non-convex angle of a high-cost region and R_P is exiting the region through P, sub-wedge W_M is not empty. The same is true when P is a convex vertex of a high-cost region and R_P is entering the high-cost region through P. In all other cases, sub-wedge W_M is empty. These results are supported by Lemmas 6, 7, 8 and 9. Thus, paths to high-cost region vertices can give rise to three new sub-wedges, as illustrated in Figure 108. (Note that Figure 108 does not include Snell's-law paths to vertices P_1 or P_2 of the high-cost region. In the situation illustrated, neither of these

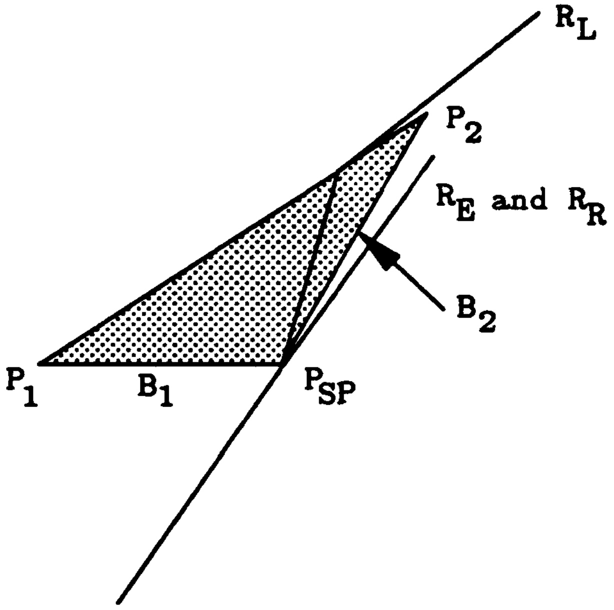
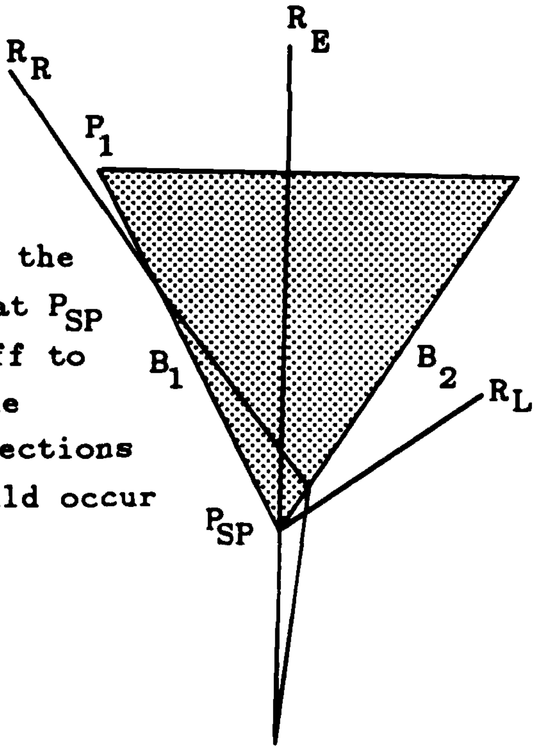


Figure 106. Intersecting Only One Side



Note: R_R should intersect the high cost region exactly at P_{SP} but the figure shows it off to the right to illustrate the intersections. Both intersections of R_R (with B_1 and B_2) would occur exactly at P_{SP}.

Figure 107. Intersecting Both Boundaries

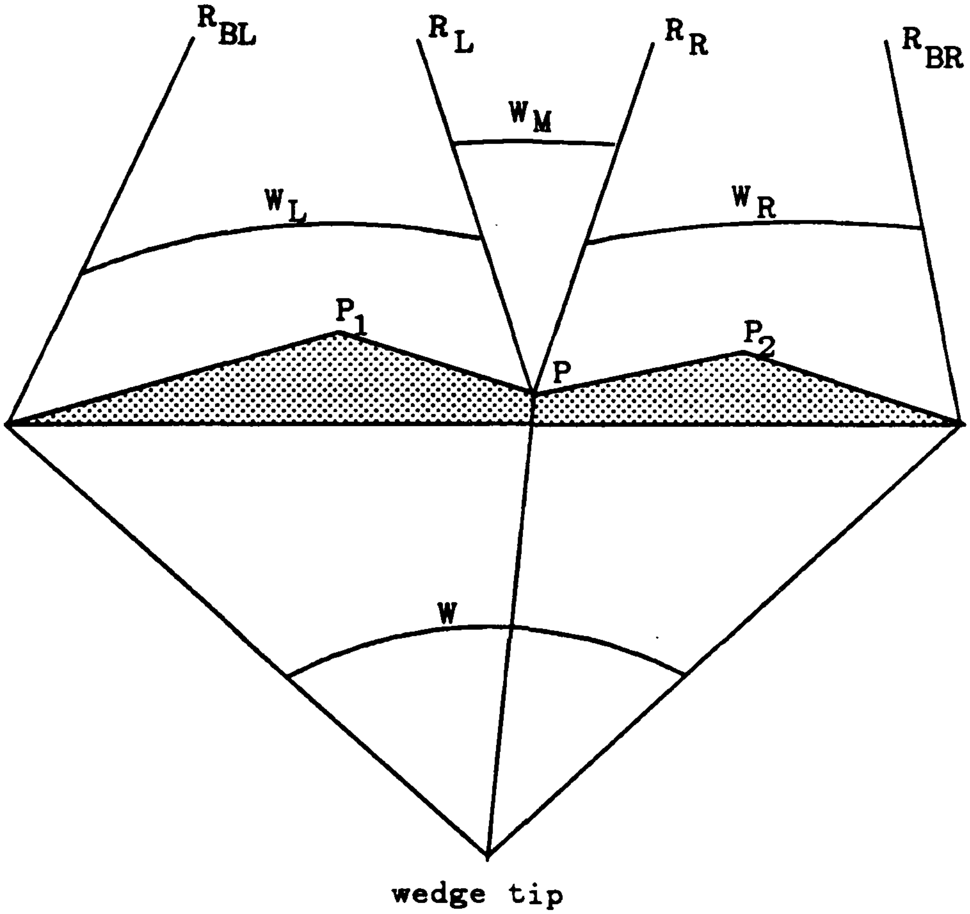


Figure 108. Refinement Into Three Sub-Wedges

region vertices has qualified as a closest unsolved search point (within wedge W). After the refinement of W, Snell's-law paths to P_1 and P_2 could be used to refine sub-wedges W_L and W_R respectively.)

When P is an obstacle region vertex, the only paths that are subjected to the splitting operation are those that do not intersect either of the two region boundaries associated with P. Recall that, by definition, Snell's-law paths terminate when they intersect an obstacle boundary. When both R_L and R_R intersect the obstacle region, the original wedge can only be refined into two new wedges, based solely on P. Figure 109 depicts this type of wedge refinement where sub-wedges W_L and W_R are refinements of the original wedge, W. When either R_L or R_R bypasses the obstacle region, three new sub-wedges can be formed, as illustrated by W_L, W_M and W_R in Figure 110.

When P is the global goal, there is generally no need to split the R_P path. By the time that the global goal becomes the closest unsolved search point, no further refinement of the wedge is, in general, profitable. However, when the goal is embedded inside a high-cost region, there can be several paths to the goal within the same wedge. Some of these paths involve critical-angle reflections, as discussed below in Section V.H. At this point in the discussion, it is sufficient to note that a path to the global goal need not be split for wedge refinement. However, when the goal is embedded, the wedge that contains it may be the subject of further search. To facilitate this, the path from the wedge tip to the goal is continued until it terminates and the wedge is refined into two sub-wedges. Figure 111 illustrates this type of wedge refinement. Figure 111(a) shows only the path

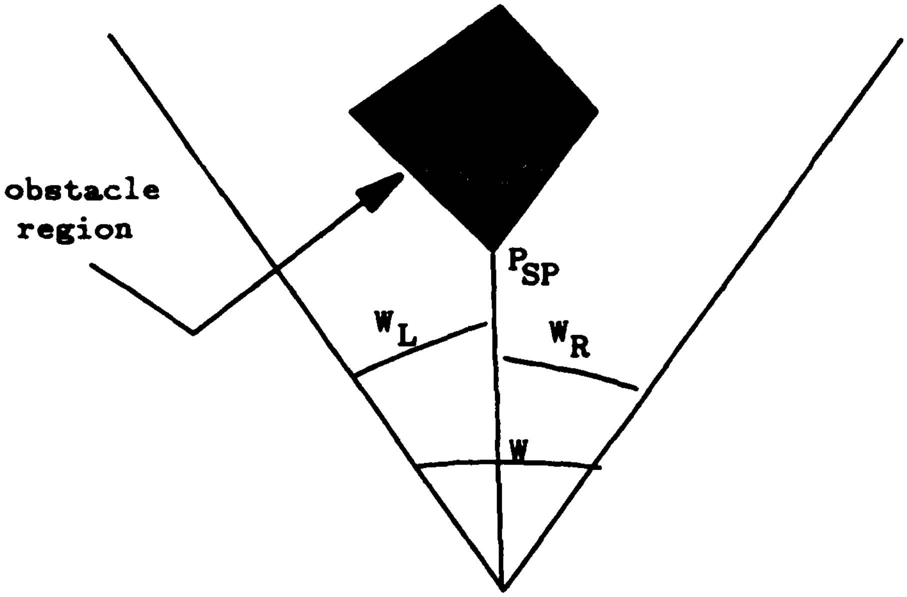


Figure 109. Refinement Into 2 Sub-wedges

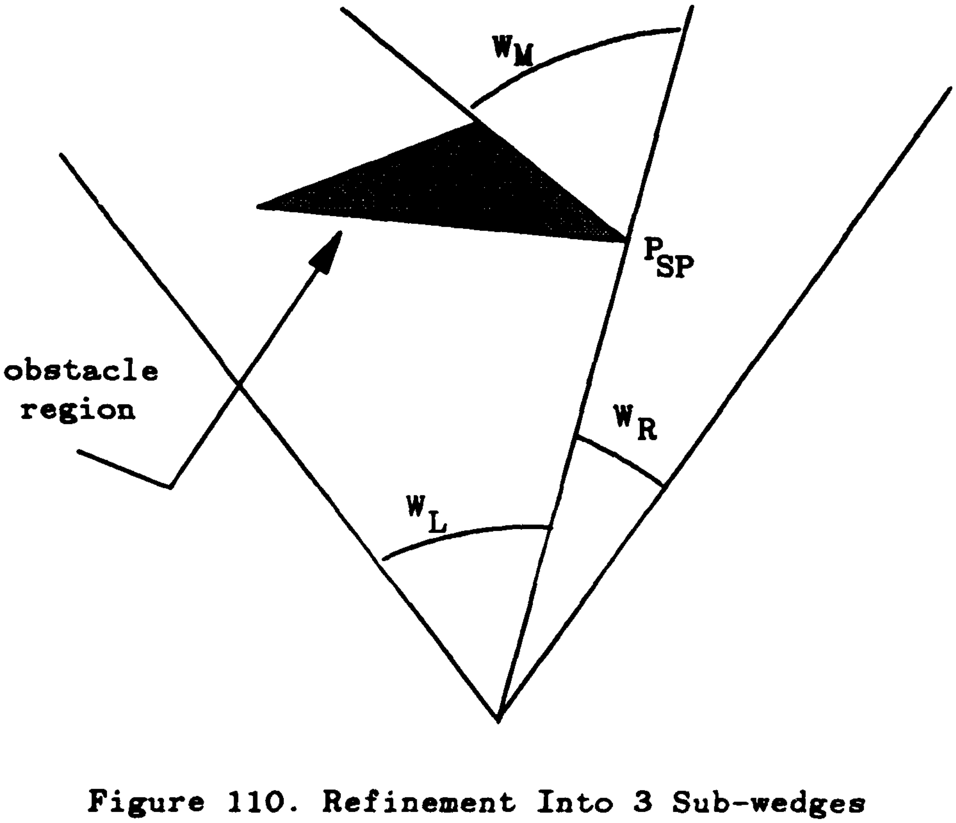


Figure 110. Refinement Into 3 Sub-wedges

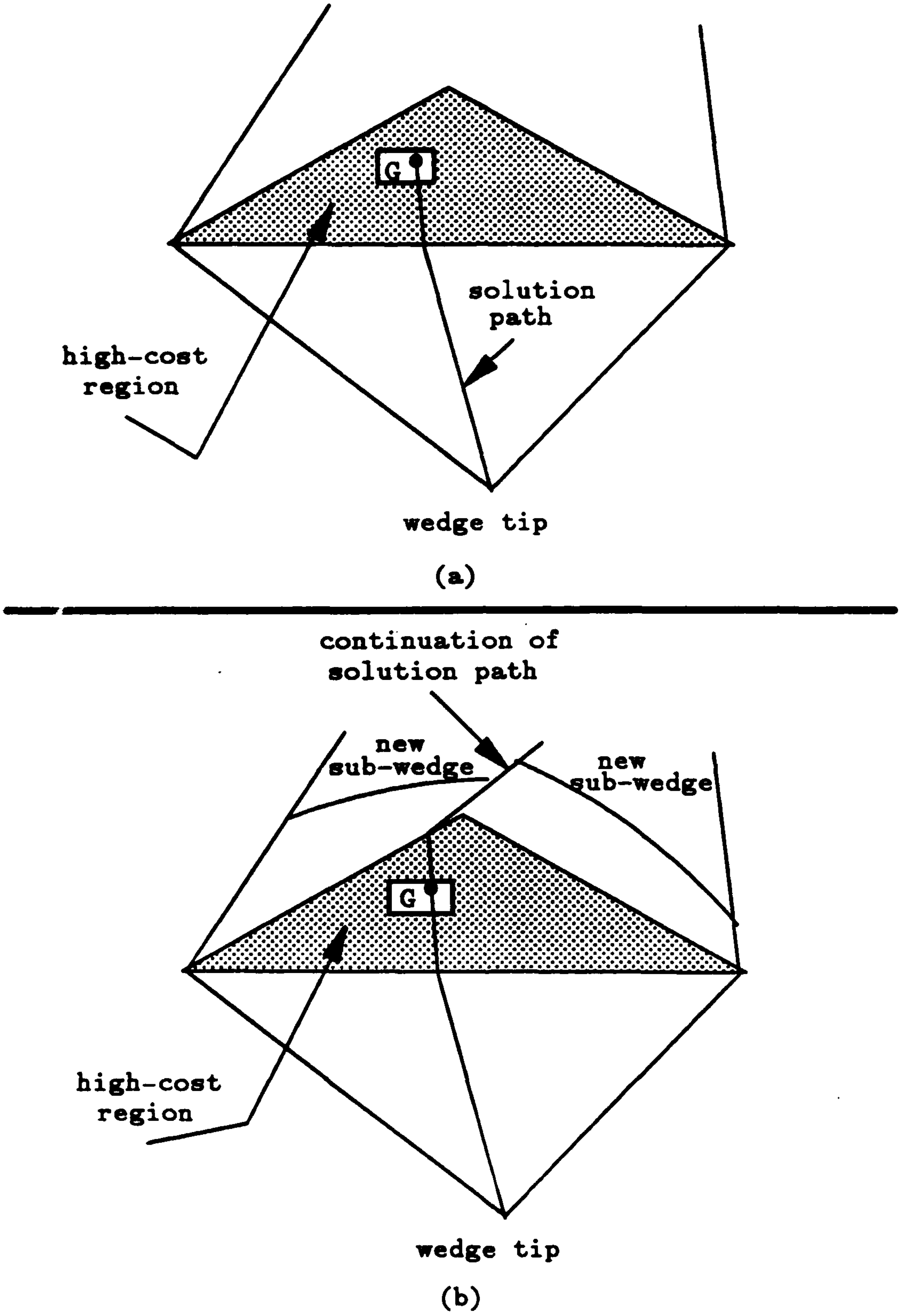


Figure 111. Refinement Based On An Embedded Goal

from the wedge tip to the embedded goal within the original wedge. (Note that the box around G in Figure 111 does not depict a cost region; it serves to allow the letter "G" to be read. A similar scheme is used later in Figures 112(a) and (b).) Figure 111(b) depicts the refinement of the wedge based on continuation of the path from the wedge tip to the goal G.

One other consideration is important when P is the global goal. In this case, R_p is a feasible solution path that has a computable cost. This cost is an upper bound on the cost of the optimal-cost solution path. Whenever the cost of R_p is less than the current upper-bound cost, it becomes the new (less costly) upper bound and R_p is retained as a possible solution.

G. ADDING NEW WEDGES TO THE AGENDA

Wedges are maintained on an ordered agenda so that the wedge having the lowest lower-bound cost evaluation is the first element of the agenda. The lemmas developed in Chapter IV are used to rate the wedges. Either point-to-point evaluations (from the wedge tip to the goal for 0-explored wedges) or boundary-to-point evaluations (from the Kth boundary to the goal for K-explored wedges) added to the cost of the minimum-cost path through a K-explored wedge can be used. If a wedge has an approach path then, by definition, any other path within the wedge must use that approach path and incur its cost. This cost is known and can be added into the evaluation. Given that such lower-bound evaluations can be made, it is a simple matter to maintain the agenda as an ordered list.

Care must be exercised when adding new wedges onto the agenda to ensure cycle avoidance. The agenda must not contain duplicate copies of the same wedge. If duplicates exist, they will have identical cost evaluations and duplicates can thus be easily detected and eliminated. In the standard A* algorithm, cycle avoidance requires keeping a Closed list and checking every agenda candidate to ensure that it has not already been removed from the agenda at some previous time. A simple way to avoid duplication in our algorithm relies on the fact that duplicate wedges must be constructed from duplicate Snell's-law paths. Thus, whenever a duplicate path to any search point is computed, wedge refinement as well as agenda update are not necessary.

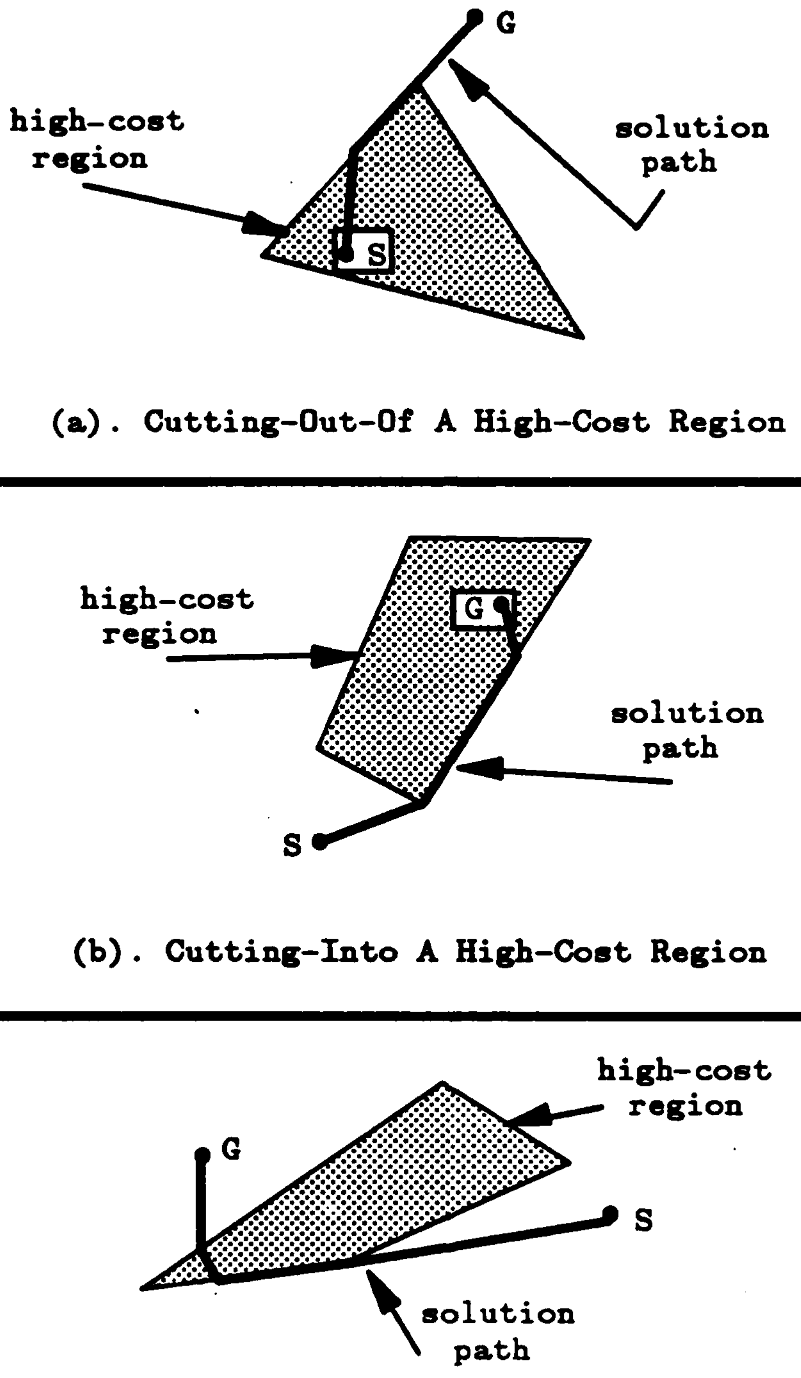
Recall that there are several other times when a new wedge need not be added to the agenda. If the wedge's lower-bound cost evaluation exceeds the current upper bound, it can be pruned. Also, if the wedge relies on an approach path to the wedge tip and another lower-cost path to the same wedge tip exists, the wedge can be pruned. A new wedge can be discarded if it is empty. Finally, Lemma 11 can be applied to eliminate new wedges.

H. TOTAL INTERNAL REFLECTIONS

The previous section completes the discussion of each box of Figure 88, the control flow of the basic algorithm. This version of the algorithm does not include considerations for all of the limitations associated with applying Snell's law to the weighted-region problem, as presented in Section IV.J. The major limitation concerns total internal reflections. Almost every routine in the algorithm must be aware of this phenomena.

Recall that the possibility of a total internal reflection can only arise when a path attempts to exit a high-cost region into a low-cost region at an angle of incidence greater than the critical angle. Here, Snell's law does not apply to the path-planning problem because adherence to the law requires that the path "double back" into the high-cost region. Clearly, such paths are not possible solution paths for a weighted-region problem. There are only three types of reflection paths that are interesting as possible portions of solution paths and all of these use exactly the critical angle as an angle of incidence. First, a path may "cut-out-of" a high-cost region at the critical angle and then travel right along the region boundary that caused the reflection. Secondly, a path may travel along a high-cost region boundary and then "cut-into" the region at the critical angle (this is a reverse path of the first case). Finally, a path may travel along a region boundary and then "cut-through" the high-cost region when its initial angle of entry into the region is the critical angle (this is a continuation of a path from the second case). A path that includes any of these critical angle reflections may be an optimal-cost solution path. Therefore, our algorithm must be able to detect and exploit such paths. Figure 112 illustrates the three cases.

When obeying Snell's law would cause a reflection to occur along a path during ray tracing subsequent to a splitting operation, our algorithm ignores the path angle of incidence with the boundary causing the reflection. Instead, we treat the path as if it had intersected the reflection-causing boundary exactly at the critical angle. This results in a path that exits the region, traveling right along the region boundary. The algorithm notes that a reflection has occurred on the path by placing a special marker in the path description (similar to a null boundary marker).



(a). Cutting-Out-Of A High-Cost Region

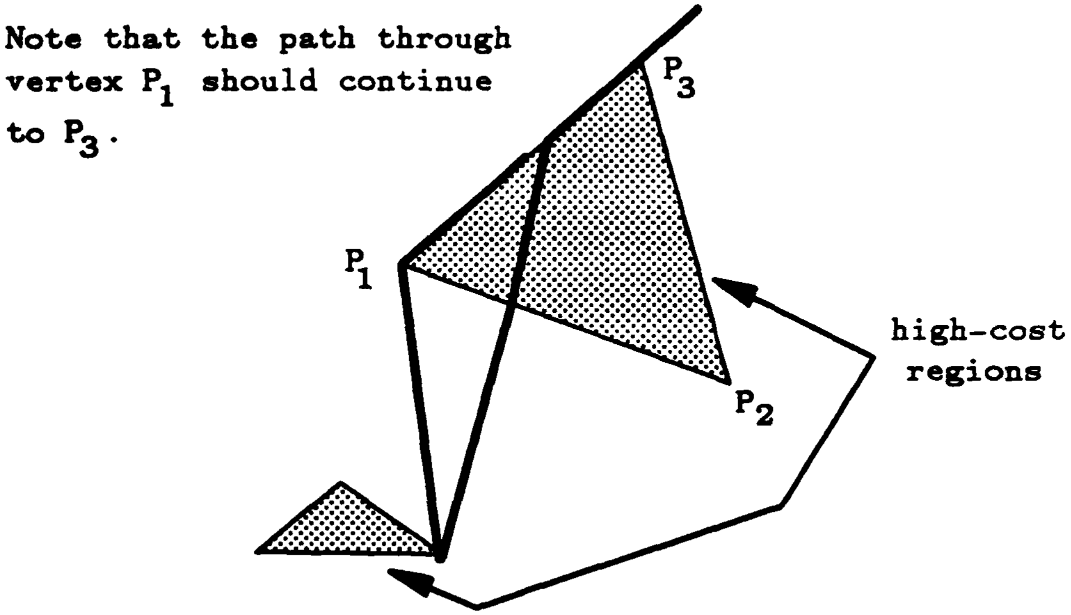
(b). Cutting-Into A High-Cost Region

(c). Cutting-Through A High-Cost Region

Figure 112. Reflection Paths Used by the Snell's-Law-Based Algorithm

This treatment (i.e., ignoring the actual angle of incidence on the path) is justified because only critical-angle reflections are interesting. If some other path exits the same boundary and also includes a reflection, the two paths together form an empty wedge. (See Figure 113.) The empty wedge will be pruned. Similarly, ray tracing need not be continued beyond the endpoint of the boundary causing the reflection. It is pointless to waste ray tracing effort on a wedge that is likely to be pruned. If the wedge is not empty (as described below), the ray tracing is completed after the critical-angle reflection path is computed.

When a second path exits the same boundary but does not reflect there must be some point on the boundary that results in exactly a critical-angle reflection. This is illustrated in Figure 114. Here, the wedge is not empty since it contains, as a minimum, the point P_3. The $V-A-B-P_3$ path is marked as a reflection path (during ray tracing). In this case, the critical-angle reflection path across boundary B_1 to point P_3 can be found without resorting to an iterative search. Instead of starting at the wedge tip, start a hypothetical path at boundary B_1 so that it makes the critical angle with the normal to B_1 as it enters the high-cost region traveling in the direction towards the wedge tip. Trace this path back through the boundaries between the wedge tip and B_1 and note the angle θ at which the path exits the last boundary before the wedge tip (boundary B_2 in Figure 114). The angles at each boundary-crossing episode along this path are invariant with lateral displacement of the path itself. Thus, to create a critical-angle reflection path (not based on iterative search) from the wedge tip to point P_3, construct a path that starts at the wedge tip and makes angle θ at its first boundary-crossing episode. This backwards-forwards tracing operation results in the $V-C-D-P_3$ path of



Note that the path through vertex P_1 should continue to P_3.

Figure 113. Two Reflection Paths Create An Empty Wedge

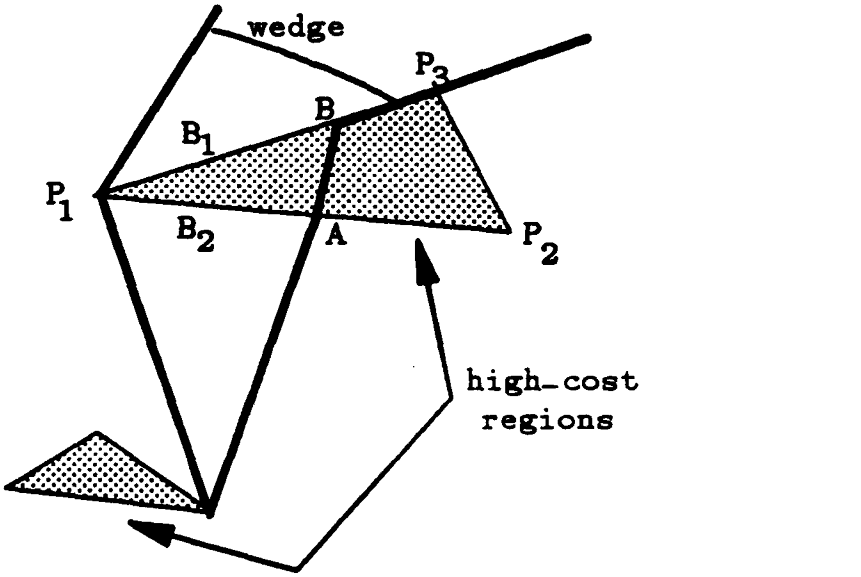


Figure 114. A Non-Empty Wedge With One Reflection Boundary

Figure 115, a critical-angle reflection path between V and P_3. (The rectangle in Figure 115 that contains "B_1" is not meant to be a cost region.) Note that the wedge formed by paths $V-C-D-P_3$ and $V-A-B-P_3$ is empty and need not be created. Also, ray tracing must be applied to the $V-C-D-P_3$ path to complete a wedge-boundary description.

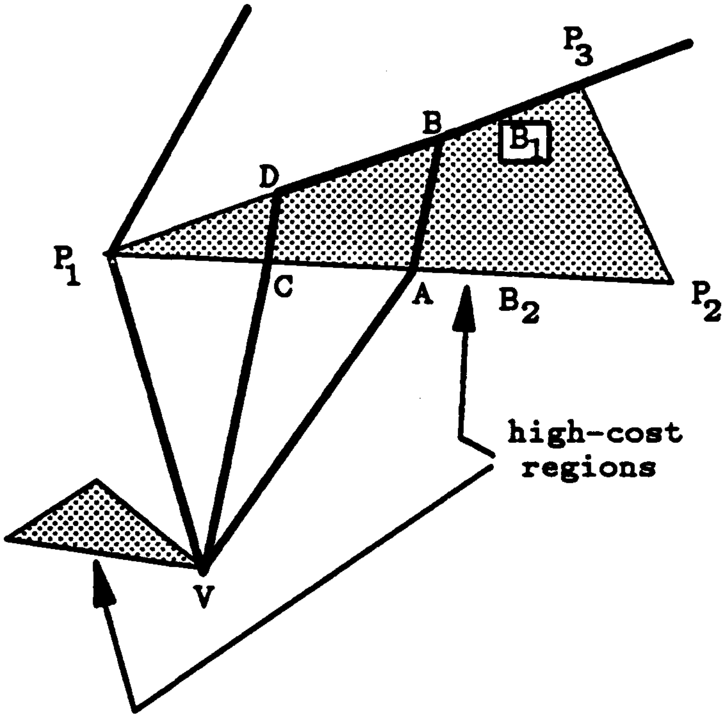


Figure 115. Finding The Critical-Angle Reflection Path

Paths beginning at wedge tips located inside high-cost regions must also consider critical-angle reflection turns. We illustrate this case in Figure 116. (Note that Figures 116 and 117 contain text boxes inside the high-cost regions; these boxes do not denote region boundaries.) Suppose that S is embedded in a high-cost region and that point P is the closest unsolved search point within wedge W.

Clearly, a straight-line path from S to P is a feasible solution. However, critical-angle reflection paths using sides B_1 or B_2 of the high-cost region, when they exist, have lower path cost. Let R_1 and R_2 denote such paths as illustrated in Figure 117. Both R_1 and R_2 are easily constructed by creating paths that intersect the appropriate boundary at the critical angle. One of the paths $S-P$, $S-P_1-P$, or $S-P_2-P$ has least cost and can be chosen at the best S-to-P path. This choice affects the refinement of the original wedge (wedge W of Figure 116) formed by paths R_L and R_R.

Up to this point in the discussion, there has been only one Snell's-law path to split when refining a wedge. When the wedge tip is embedded in a high-cost region, as many as three such paths are available on which to base wedge refinement. (Note that two of these paths are the "cut-out" paths as in Figure 112.) Recall that the purpose of refining a wedge is to define those boundaries that are intersected by other paths through the sub-wedges and to define the limits of the sub-wedges. Again, refer to Figure 117. If a critical-angle reflection path across side B_2 exists, then the left side sub-wedge should be based on the reflection path. Specifically, the left side sub-wedge must be based on the R_L path and the $S-P_2-P$ path. If side B_2 does not allow a critical-angle reflection path to exist, then the left sub-wedge should have the path resulting from ray tracing the $S-P$ segment as its right boundary-defining path. Similar statements hold regarding the existence of a reflection path across side B_1 of the high-cost region and the right side refinement (sub-wedge) of W.

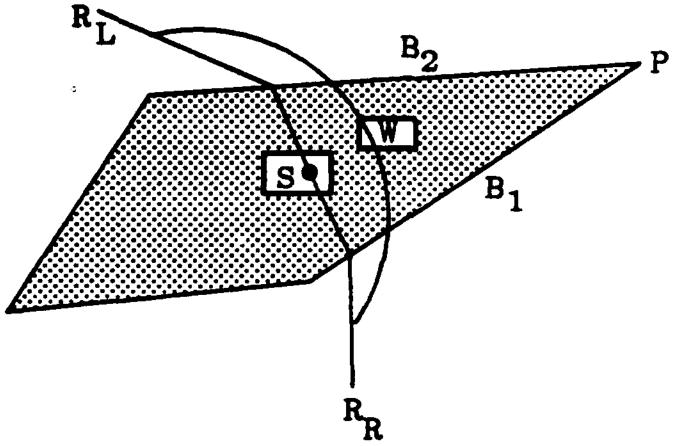


Figure 116. An Embedded Start, S

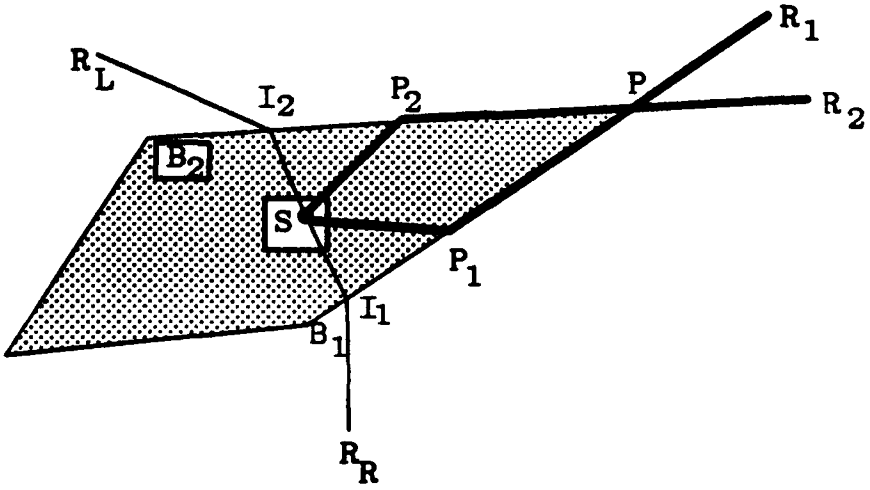


Figure 117. Reflection Paths From An Embedded Start

When the wedge tip is inside a high-cost region, the refinement of the middle sub-wedge must be treated somewhat differently. Clearly, the least-cost path of the three possible S-to-P paths must be used as the approach path for the middle wedge. Suppose, in Figure 117, that $S-P_1-P-R_1$ is the least-cost path S-to-P path. Then the middle wedge must be formed by $S-P_1-P-R_1$ as a right boundary and the continuation of the $S-P_1-P-I_2$ path as a left boundary. Similarly, if $S-P_2-P-R_2$ is the least-cost S-to-P path, the middle wedge is defined by $S-P_2-P-R_2$ on its left and the continuation of the path $S-P_2-P-I_1$ on its right. When the path $S-P$ is the least-cost path, neither side B_1 nor B_2 allows a critical-angle reflection path to exist and the middle wedge can be normally created, splitting the $S-P$ path at vertex P.

The existence of critical-angle reflection paths also affects the addition of new wedges to the agenda. Whenever a critical-angle reflection path to a vertex is located, there is an opportunity to create "cutting-into" and "cutting-through" paths. A reflection wedge can be created from such paths, as illustrated in Figure 118. Suppose that R_{REF} is a path that travels right along side B_1 of a high-cost region and includes the subpath $S-P_1-P_2$. A new path could enter the high-cost region at the critical angle anywhere along side B_1, between points P_1 and P_2. Thus, we can create a new wedge defined on its left by R_L and on its right by R_R. This wedge has P_1 as a wedge tip. Path R_L enters the high-cost region at the critical angle through point P_1, similarly with path R_R and point P_2. Recall that the angle of incidence that a path makes with a region boundary during a boundary-crossing episode is invariant with displacement of the intersection point.

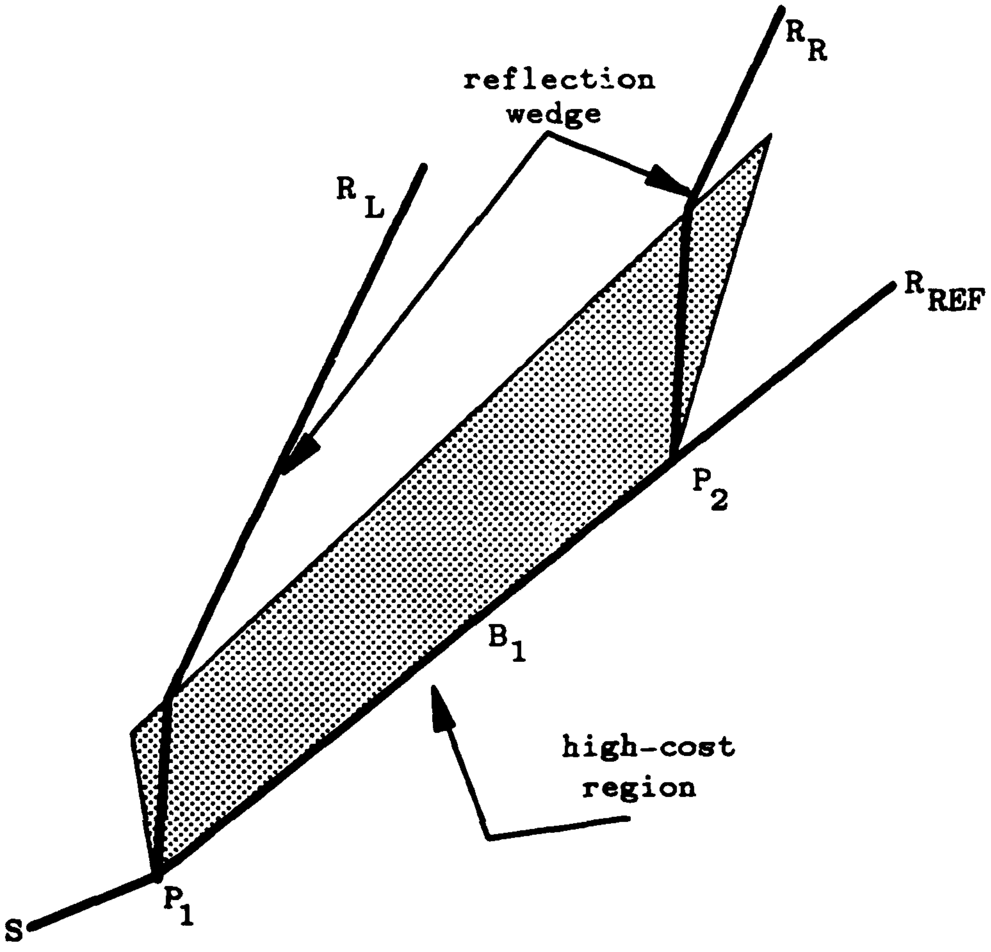
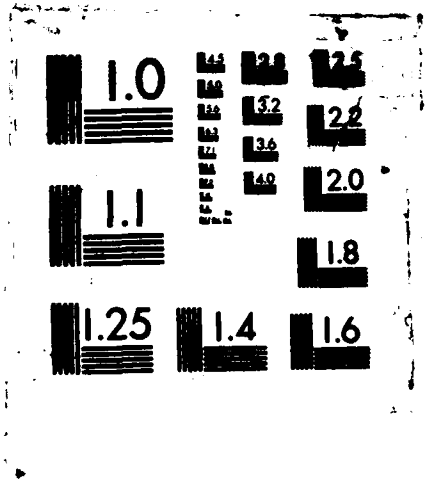


Figure 118. A Reflection Wedge

AD-A183 900 SOLVING A CLASS OF SPATIAL REASONING PROBLEMS: 4/5
MINIMAL-COST PATH PLANNING IN THE CARTESIAN PLANE(U)
NAVAL POSTGRADUATE SCHOOL MONTEREY CA R F RICHBOURG
UNCLASSIFIED JUN 87 F/G 12/9 NL



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Thus, when R_L and R_R form a partial well-behaved Snell's-law path pair (K-WBSP) for K > 1, path R_L runs parallel to path R_R through the first K boundary-crossing episodes. Thus, there is no need for iterative search within reflection wedges. Instead, we find the lateral displacement that causes either R_L or R_R to travel through the desired search point. The solution path will have this displacement within the reflection wedge. Note that this analysis assumes that the solution path (up to the search point), R_L and R_R all intersect the same sequence of region boundaries. This behavior is guaranteed since refinement is based on a Snell's-law path to the *closest* unsolved search point. If different region boundaries could be intersected by the three Snell's-law paths, they may not be piecewise parallel to each other.

Since a reflection wedge does not require an iterative search to find a point-to-point solution path, these wedges must be somehow identified when added to the agenda. We chose to maintain a separate agenda for reflection wedges, processing this agenda after the regular agenda is emptied. The reason for this choice is that reflection wedges can be numerous and delaying their processing should result in a low upper bound on the cost of the optimal solution path, enhancing pruning.

Recall that the algorithm requires the refinement (i.e., creation of sub-wedges) of parent wedges based on a path to the global goal only if the goal is embedded inside a high-cost region. This requirement is based on the existence of cutting-in paths. Suppose that a goal path has been found within wedge W, as shown in Figure 119(a). (In the Figure, the boxes around "Goal" and "B" do not mark region boundaries; they only serve to keep the letters from being shaded.

Similarly in Figure 120 with the letters "S" and "G".) Note that the wedge right-side-defining path, R_R, nearly reflects after intersecting side B of the high-cost region. Continuing the wedge tip-to-goal path through side B of the region results in a reflection path, R_{REF}, which together with path R_R forms a new sub-wedge as shown in Figure 119(b). At some later time, this wedge may be removed from the regular agenda to find the exact critical-angle reflection path to vertex P_1. At that time, a cut-in path segment from P_2, across side B to the goal can be found. If the original goal path of Figure 119(a) had not been continued and used to refine wedge W, this cut-in path would not have been noticed.

Reflection wedges can also be refined. The left and right sub-wedges, as usual, have the same wedge tip as the parent wedge. Thus, these two sub-wedges are also reflection wedges and should be either pruned or added to the reflection agenda. The middle sub-wedge, when it exists, has a new wedge tip and is not, in general, a reflection wedge. These sub-wedges should be treated as regular wedges that happen to include a reflection path as an approach path. Note that all the wedge-pruning criteria previously developed also apply to reflection wedges.

We have described methods to ensure that cutting-in, cutting-out and cutting-through paths are considered. Combinations of these paths are also possible. Figure 120 depicts a cutting-out cutting-in combination. The algorithm solves this case by first creating those wedges associated with the embedded start, S. One sub-wedge has the reflection path across boundary B as a left-side-defining path. This wedge leads to the creation of the reflection wedge, between point P_R and P, cutting into the region across boundary B. This reflection wedge contains goal G, and a cutting-in path to G is found.

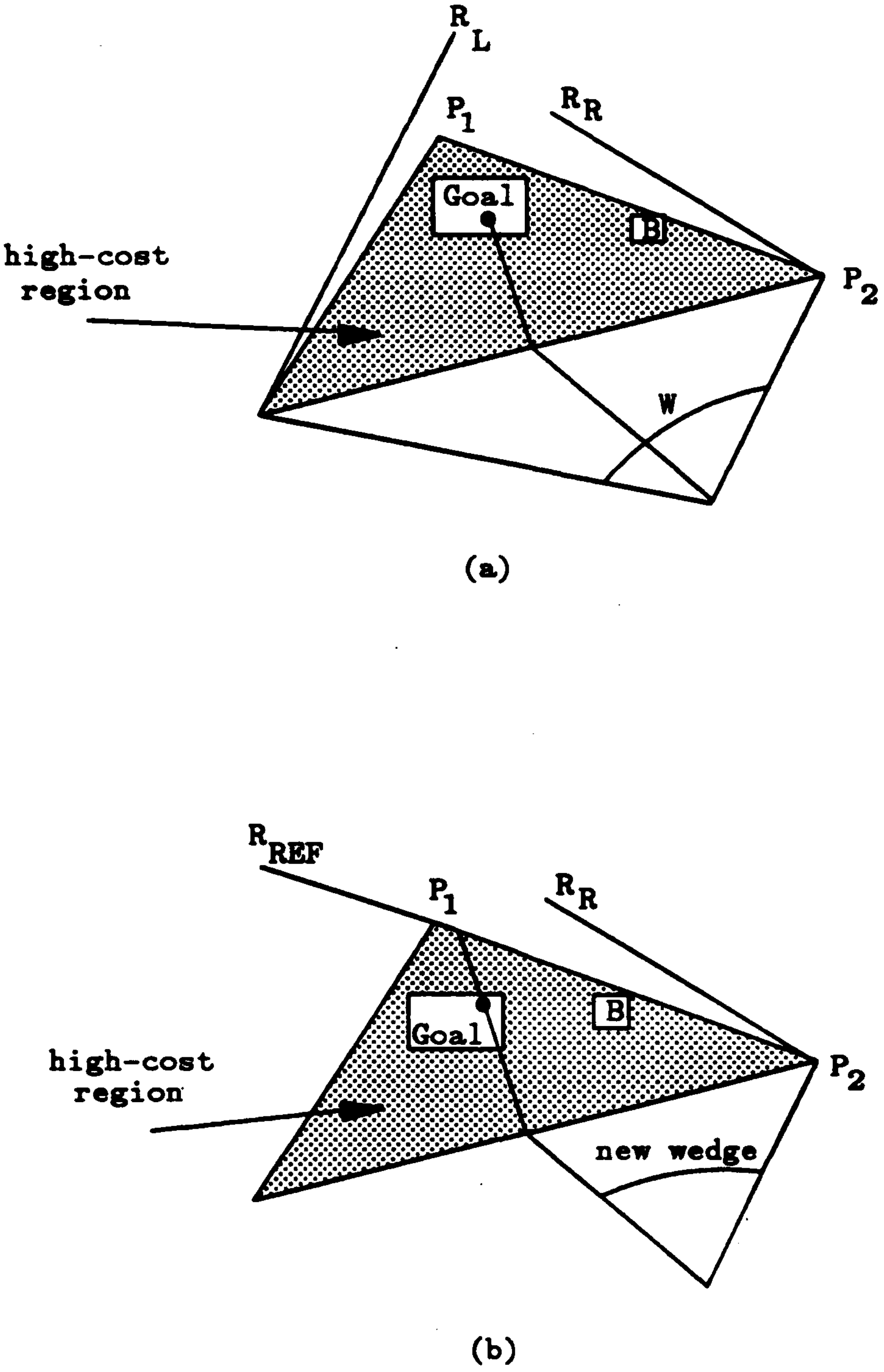


Figure 119. Refinement Based On A Solution Path

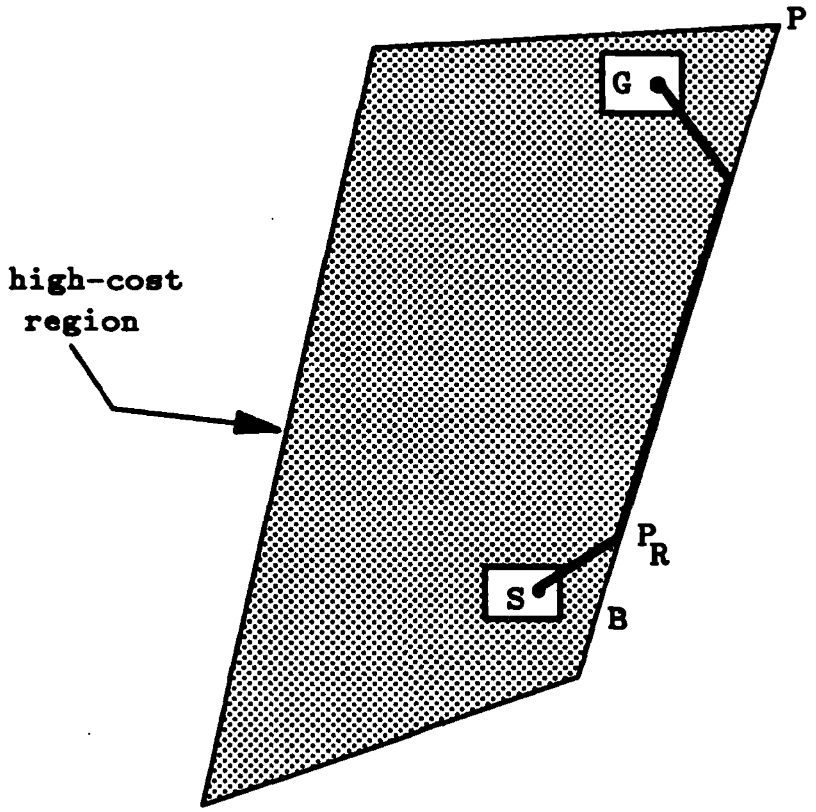


Figure 120. Cutting-Out And Cutting-In

I. BLIND REGIONS AND LOCALITY ASPECTS

Chapter IV presented several other limitations, in addition to total internal reflections, that can occur when applying Snell's law to the weighted-region problem. Those that have not yet been discussed center on the existence of blind regions and the localized nature of the law. Both of these limitations are treated by the algorithm without further extension.

Blind regions feature a diffraction vertex at their base. These vertices are simply treated as new starting locations that have a known approach path. The wedge forming the blind region serves the same purpose as the initial wedge does for the start location. Therefore, the algorithm can be recursively applied to diffraction vertices. The localized nature of Snell's law cannot be explicitly overcome. We rely on good pruning heuristics to avoid "looking everywhere" and thus ensure some degree of efficiency.

J. REDEFINING THE ALGORITHM

We have discussed enhanced capabilities for the algorithm since its initial presentation in the introduction to this chapter. In Figure 121, we provide a more detailed view of the control flow for the algorithm that includes provisions for the concepts discussed in preceding sections. Notably, ability to work with total internal reflections is added.

The initialization procedures required are the same as presented in Figure 88. There is new decision box required to refine a wedge based on a path to the global goal when the goal is embedded in a high-cost region. Such refinement allows

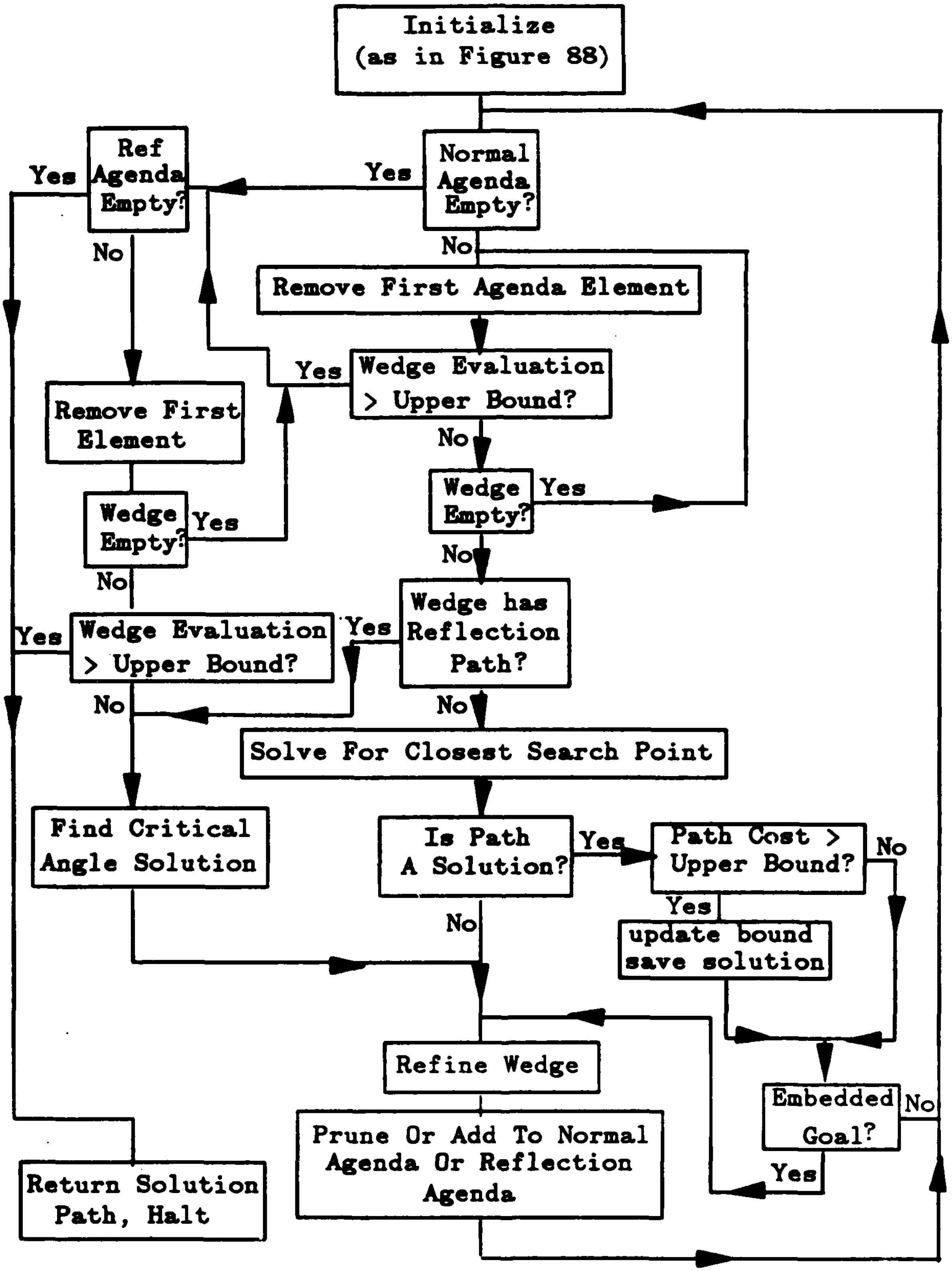


Figure 121. Redefined Algorithm Control Flow

"cutting-in" paths to the goal to be computed. The diagram in Figure 121 also shows an entirely new branch of operations to be completed when the regular agenda is exhausted. This branch of control processes reflection wedges. Note that the algorithm prefers to process regular wedges rather than reflection wedges. This is an effort to achieve the lowest possible upper bound on the cost of a solution path so that the maximum number of reflection wedges can be pruned without search. In general, a large number of reflection wedges are created and many of them can "double-back" on themselves. We hope to eliminate such wedges by pruning based on a low upper bound. Also, the new algorithm control-flow scheme requires that both the regular agenda and the reflection agenda are empty or have only elements whose cost evaluations exceed the upper bound on the cost of the optimal solution path before processing is terminated.

K. DEMONSTRATION

In this section, we present a demonstration of the algorithm as described in Figure 121. The demonstration problem is not complicated, involving only 2 high-cost regions and no obstacle areas. The cost-rate ratio is 2:1 between high- and low-cost regions. In Figure 122, S denotes the start location, G the goal location and the shaded triangles represent high-cost regions. The bounding box is displayed as the heavy-lined rectangle that intersects the bottom high-cost region. The intersections of the high-cost regions and the bounding box required the creation of (artificial) boundary points z7 and z9. (All path intersections with the bounding box are denoted z*X* where *X* is a number.)

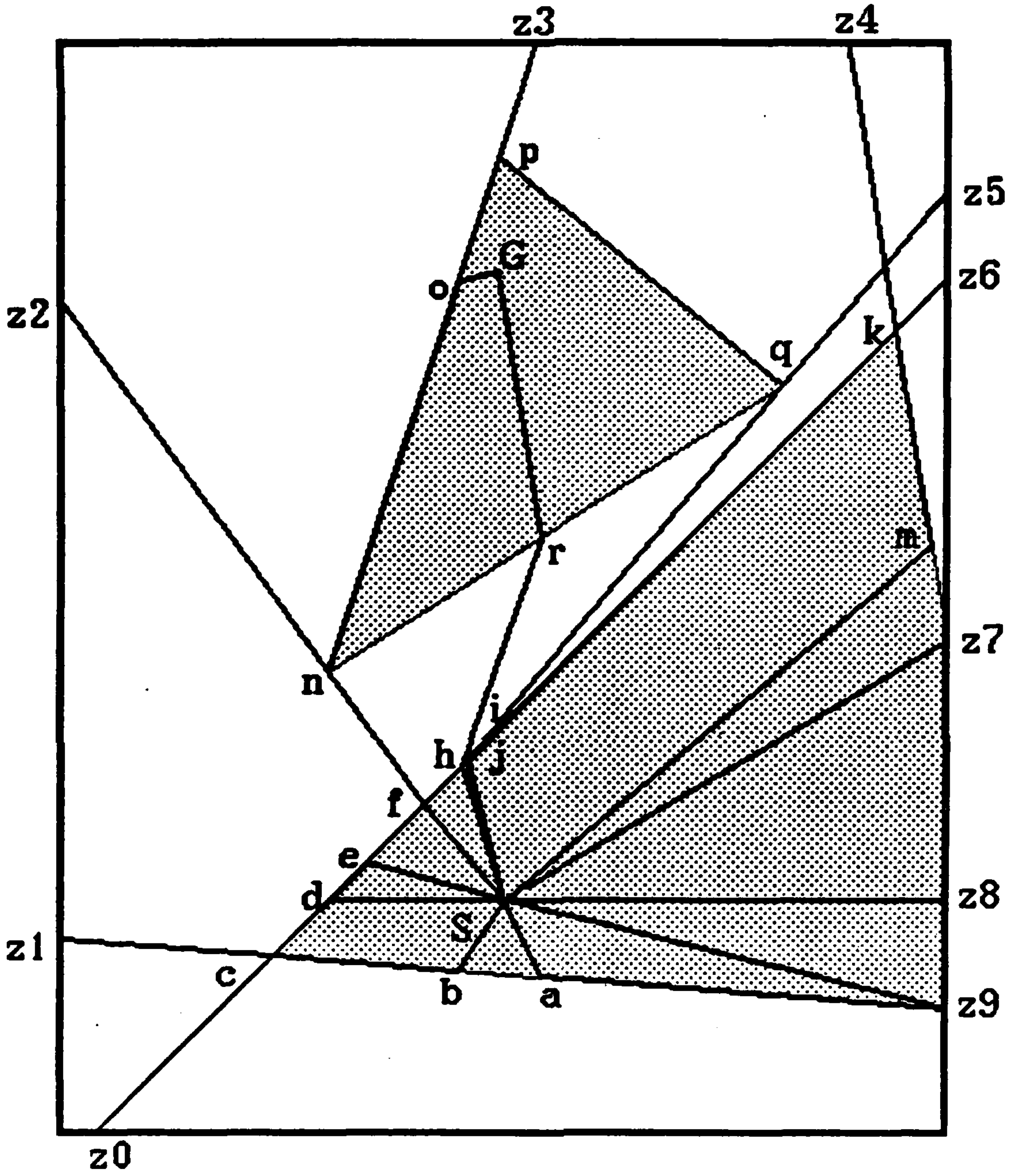


Figure 122. Demonstration Problem

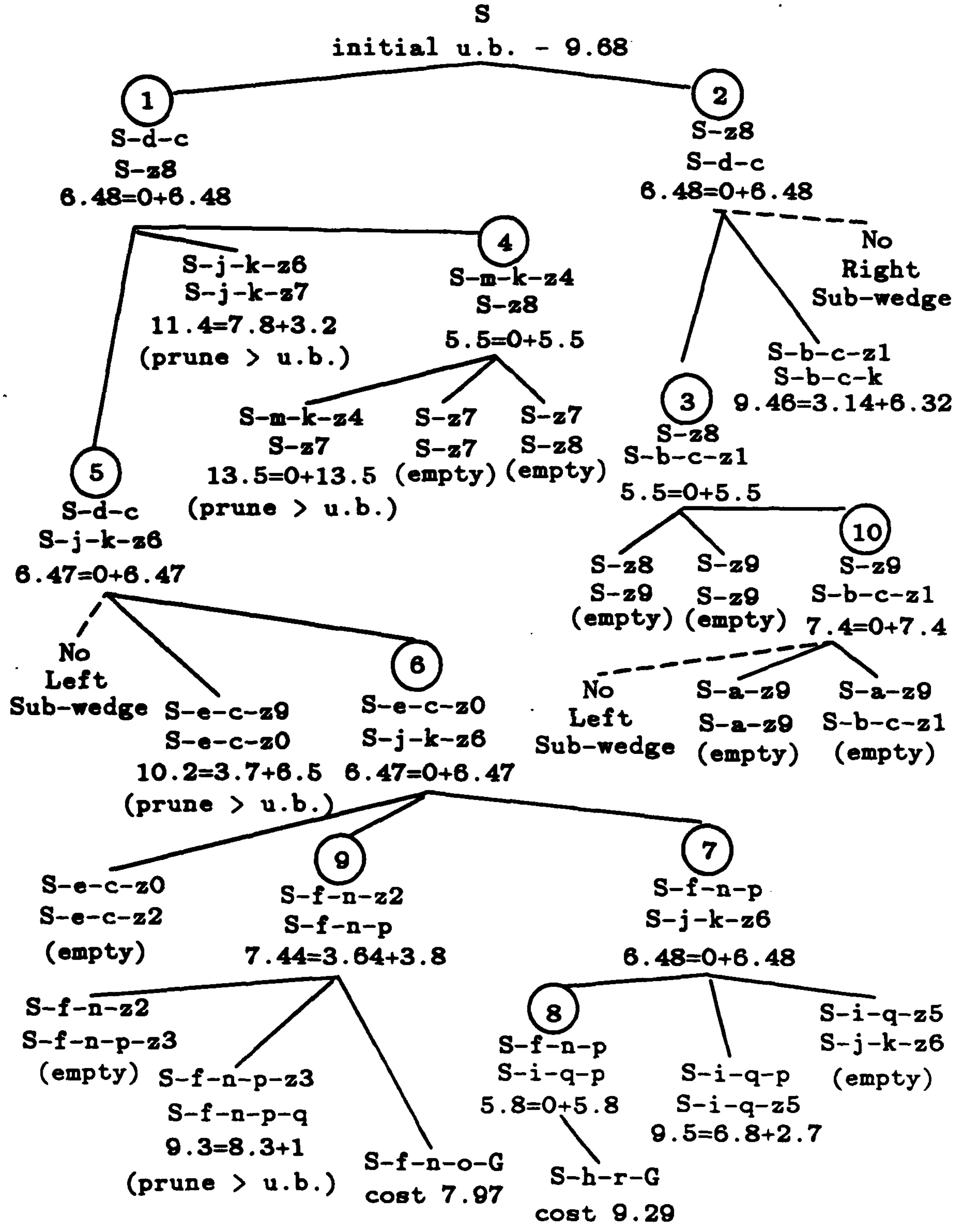


Figure 123. Search Tree

Figure 123 details the search tree for the problem. This is a tree of wedges. At each node, the Snell's-law path defining the wedge left side is listed above the path defining the wedge right side. The descendants (sub-wedges) of each node (parent wedge) are listed from left to right as the left, middle, and right sub-wedge. The circled number above each node indicates the order in which the node was removed from the agenda for refinement. If a node has no number above it, it was either never added to the agenda or it remained on the agenda at solution. Some nodes also include their numerical rating in the form f(n)=g(n)+h(n) (the total cost evaluation is equal to any known cost added to a lower-bound cost evaluation). The approach path constitutes a known cost. In the figure, the abbreviation "u.b." denotes upper bound.

The search begins with two initial wedges, created by ray-tracing Snell's-law paths that have initial headings perpendicular to the S-to-G line segment. The right such path proceeds from S with no boundary-crossing episodes until the bounding box is intersected at point z8. The left initial path reflects at point d, its first boundary-crossing episode. Note that this path is not a critical-angle reflection path, although it is treated as such for the time being. Tracing of this path is terminated at point c. Early ray tracing termination is possible here because if a wedge associated with the S-d-c path is not empty, a critical-angle reflection path to point c will be found. Ray tracing will be completed at that time.

The first wedge examined is the upper initial wedge. Here, point k is chosen as the closest unsolved search point (although point c or z7 could also have been selected). The algorithm locates two reflection paths from S to k, S-j-k and S-m-k.

The S-j-k path has lower cost, so it is used in the left and middle sub-wedges while the S-m-k path must be used for the right sub-wedge. Next, the algorithm considers the lower initial wedge that is defined by S-z8 on its left and S-d-c on its right. Point c is selected as the closest unsolved search point. Note that no right side sub-wedge is created during refinement of the lower initial wedge. There is no need for this sub-wedge since the left side sub-wedge includes all the interesting space within the parent wedge. The middle wedge has a critical-angle reflection path to point c as an approach path. This middle sub-wedge remains on the agenda at solution and is never further refined.

Processing continues in a similar manner until the eighth wedge is refined. This wedge yields a refraction path to G that features two boundary-crossing episodes. This path has a cost slightly lower than the cost of the straight-line path used as the initial feasible solution path and thus causes replacement of the upper bound on the cost of the optimal solution path. The next wedge refined also produces a solution path. This is a critical-angle reflection path to G that quickly exits the first high-cost region at point f, travels along the region boundary between points n and p, and then cuts-into the second high-cost region at point o to reach the goal. This path, S-f-n-o-G turns out to be the optimal-cost solution path.

After the tenth wedge is examined. the regular-wedge agenda is exhausted. all of its elements having cost greater than the cost of the S-f-n-o-G path. All elements on the reflection-wedge agenda also exceed the upper bound so processing terminates. Note that the wedges having approach paths S-i-q and S-b-c were never examined evcn though both contain cutting-in paths to the goal. (The former has a cut-in path from the boundary between points q and p while the latter contains the path S-b-c-n-o-G.) Both wedges were pruned due to the upper-bound cost established after refinement of the ninth wedge.

L. COMPARISON WITH THE CONTINUOUS DIJKSTRA ALGORITHM

Both the Continuous Dijkstra Algorithm (CDA, discussed in Section II.F.5) and the algorithm developed in this chapter rely on using a homogeneous-cost region problem representation and applying Snell's law as a local optimality criteria to solve specific instances of the weighted-region prcblem. Thus, both algorithms rely on the same general precepts. However, there are many differences in the two algorithms.

A primary objective in developing the CDA was to establish polynomial bounds for the time and space complexity of the algorithm [Ref. 42]. Many of the control flow decisions made during algorithm design were based on achieving this objective. Our Snell's-law-based algorithm is intended to support low average-case time and space requirements and the algorithm has been designed accordingly. Thus, the two algorithms are based on different design goals and these design goals affect the control-flow scheme for both algorithms.

The CDA uses Dijkstra's algorithm (uniform-cost search) which is an uninformed strategy. In this chapter, we have developed many criteria that can be used to prune nodes (i.e., wedges) in the search tree that we create. We have also developed methods to achieve lower-bound cost evaluations for each node in the search tree. These two developments lead to the use of the A* algorithm, an informed strategy which offers improved average-case performance over uninformed strategies when reliable lower-bound evaluation functions are available. Thus, our Snell's-law-based algorithm focuses, to a much greater degree, on keeping the search tree small in order to minimize search effort.

The CDA uses uniform-cost search to find wedges (all of which begin at the start) that define disjoint intervals of optimality for each homogeneous-cost region boundary on the area-cost map. There is a single wedge associated with each such interval so that the optimal-cost path from the start to the portion of the region boundary (i.e., the interval) must be within only that wedge. The collection of intervals of optimality amounts to an exhaustive graph. Thus, a specific start-to-goal path planning problem can be resolved by locating the single interval of optimality that contains the goal. In some sense, the CDA uses the start (location) in a pre-processing step to create an exhaustive graph that simplifies finding a specific start-to-goal optimal-cost path. In contrast, our Snell's-law-based algorithm focuses on finding the optimal-cost start-to-goal path immediately and only creates a graph (a tree) structure large enough to find that path. That is, only the minimum number of optimal-cost paths are found during execution of the algorithm. Again, this choice is influenced by our goal of achieving low-cost average-case performance.

There are other differences between the two algorithms. The CDA must triangularize the homogeneous-cost regions, resulting in the creation of artificial region boundaries (and this increased number of boundaries affects the time and space complexity of the CDA). Our Snell's-law-based algorithm does not require triangularization; it can reason directly about homogeneous-cost regions having arbitrary geometry. No artificial boundaries are created. The CDA only solves problems for starts and goals that are located on region boundaries. Our Snell's-law-based algorithm can find the least-cost path between any two points that are not inside obstacle regions. Thus, if an optimal-cost path on the area-cost map exists, our Snell's-law-based algorithm can find that path. The CDA has polynomial time and space bounds. The worst-case bounds for our Snell's-law-based algorithm are much higher (see Section VI.B), although the average-case performance of the algorithm seems to be a quadratic function of the number of region vertices inside the bounding box (as developed in Chapter VII). Thus, the two algorithms rely on common precepts but have fundamentally different capabilities and operational characteristics.

M. SUMMARY

Snell's law can be used as the basis for a weighted-region problem solution technique. The solution path in Figure 122 shows that the method is not affected by problems of digital bias. Paths are described by a set of turn points. The difficulties associated with the law, notably those involving total internal reflections, are not insurmountable. The algorithm is able to make use of critical-angle reflection paths as well as normal refraction paths. Various pruning heuristics are necessary to overcome the localized nature of Snell's law and thus provide for some degree of efficiency. The average-case time and space performance of the algorithm is the subject of the next chapter. In the introduction, we cited the ability of this algorithm to provide improved average-case time and space performance when compared to competing techniques. We exemplify this claim in Chapter VI which presents a direct comparison between the Snell's-law-based method and the wavefront-propagation technique.

VI. PERFORMANCE COMPARISON

A. INTRODUCTION

We have implemented a version of the Snell's-law-based A* search algorithm developed in Chapter V in the C-Prolog language. A wavefront-propagation algorithm, (unidirectional ellipse-based) as described in Chapter III has also been implemented in the same language. We have used both algorithms to solve the same set of weighted-region problems in order to compare the performance of the techniques. Our effort is directed towards obtaining a notion of the average-case performance of the two methods, without regard to their worst-case complexities. It is difficult to prove that a single instance of a weighted-region problem has "average complexity. Because of this, we have presented a variety of problems to both techniques for solution.

Selecting a measure for performance comparison of the two algorithms is difficult because they have different fundamental operations. The basic operation for the wavefront-propagation algorithm is node expansion. The Snell's-law-based algorithm has no notion of lattice nodes; the algorithm uses line intersections extensively. Also, the wavefront algorithm has no concept of homogeneous-cost regions and is affected by the distance between the start and goal coordinates. The number of homogeneous-cost regions inside the bounding box (and thus the number of region vertices) is very important to the Snell's-law-based algorithm while the start-to-goal distance is immaterial. Due to the very fundamental differences between the algorithms, we base our comparison on the execution times required by the algorithms to reach a solution path and on the storage space required to describe a specific weighted-region problem to each algorithm. (We note that comparison could be based on other measures, such as the number of cpu load instructions. However, time is a well-understood concept that is also easily measured.)

Timings are achieved by the use of the built-in C-Prolog predicate "cputime" which returns the time used by the central processing unit (cpu). Both algorithms make use of the same initial solution path to limit the physical portion of the map that is searched. Because of this, the time required to achieve an initial solution path and the time required to construct a problem description is not counted in the timings for either method. We measure space requirements in terms of the amount of storage required to describe the problem after the initial bounding solution path has been found. (Prior to finding the initial solution path, the storage requirement is constant; the space to describe the entire map is needed.)

In the next section, we discuss theoretical issues relating to the Snell's-law-based algorithm. In the following sections, we provide more detailed information on the Prolog implementations of the two algorithms. Section VI.C describes the ternary-cost maps used for posing test problems as well as the manner in which test problems were selected. Performance-comparison data is presented in Section VI.D. Section VI.E briefly summarizes some results, although principal conclusions are presented in Chapter VII.

B. THEORETICAL MEASURES

As the Snell's-law-based algorithm is intended to have low time and space requirements in the average case, it is difficult to construct a meaningful worst-case problem for the algorithm. By meaningful, we indicate a problem somehow "bad enough" so that its solution requires the greatest possible time and space while, at the same time, not being "so bad" that the problem could never occur. We have not succeeded in finding such a worst-case problem and are thus forced to rely on a worst-case analysis that grossly overestimates the worst possible performance of the algorithm. The main difficulty in constructing a meaningful worst-case problem stems from the fact that the wedges we create may overlap each other. Because of this, the total number of wedges that can be created during the execution of the algorithm is difficult to tightly bound.

Before analyzing the worst-case performance, we show that the Snell's-law-based algorithm always halts. One of the first steps in the algorithm is to find an initial feasible solution path and then calculate the cost of this path. This cost is used as an upper bound on the cost of the optimal-cost solution path. Any wedge that has a lower-bound evaluation greater than this upper bound can be pruned from the search tree.

A main goal of the algorithm is to constantly extend (by refinement into sub-wedges) the portions of wedges that are explored. Basing wedge refinement on the Snell's-law path to the closest unsolved search point within the wedge guarantees this behavior. A portion of the total lower-bound evaluation for each wedge comes from the minimum cost of any path through the K-explored portion of that wedge. As the length of the explored portion of each wedge must increase, so must a portion of the lower-bound evaluation for each wedge increase. Thus, continual refinement of wedges must eventually lead to one of two possibilities. Either a start-to-goal solution path is found within a wedge and further refinement of the wedge stops, or the lower-bound evaluation for the wedge exceeds the upper bound on the cost of an optimal-cost solution path and further refinement of that wedge stops. Therefore, no wedge is subjected to infinite refinement; thus the algorithm only creates a finite number of wedges and the algorithm is guaranteed to halt.

In order to establish a worst-case time complexity for the algorithm, we must determine the maximum time that it could run before halting. Our Snell's-law-based algorithm creates a search tree where nodes in the tree correspond to wedges in the search space. Each node has a maximum branching factor of four since at most four sub-wedges can be created from the refinement of a single wedge. To see this, first consider the refinement of a regular (i.e., non-reflection) wedge. If refinement is based on a Snell's-law path to an obstacle region vertex, either two or three sub-wedges can be created (as described in Section V.F and Figures 109 and 110). When a Snell's-law path to a high-cost region vertex is the basis of refinement, the possibility for creating four sub-wedges exists. The first three sub-wedges are the left, middle and right sub-wedges that can be added to the regular-wedge agenda (as in Section V.F). In some cases, a single reflection wedge can also be created. This can occur when the wedge tip of the parent wedge (i.e., the wedge being refined) is a homogeneous-cost region vertex and the approach path to the middle sub-wedge (or child wedge) goes through another homogeneous-cost region vertex that is on the same region boundary as the wedge tip. In this case, a reflection wedge can be created based on the approach path (as described in Section V.H). Thus, each regular wedge can be refined into at most four sub-wedges.

The refinement of reflection wedges is similar. Recall that reflection wedges only exist in relation to boundaries of high-cost regions, thus obstacle regions are not involved. Here, only three sub-wedges can be created for each wedge refined. The left and right sub-wedge will also be reflection wedges. The middle sub-wedge is a regular wedge that includes a critical-angle reflection at one of the boundary-crossing episodes on its approach path. Clearly, the middle sub-wedge cannot lead to the creation of a reflection sub-wedge. Thus, nodes in the search tree that correspond to reflection wedges have a maximum branching factor of three.

Note that the refinement of any wedge is based on the Snell's-law path to the closest unsolved search point within that wedge. Further, the closest unsolved search point will be on a wedge-defining Snell's-law path for each of the sub-wedges created during refinement. Recall that when determining the closest unsolved search point within a wedge, those search points already on wedge-defining Snell's-law paths are not considered as eligible candidates.

The total number of search points is at most the number of region vertices plus the goal. Denote this number SP. Thus, each of the two initial wedges (as in Figure 90, Section V.B) cannot contain more than SP search points. (Clearly this is an overestimate since the two initial wedges do not overlap. The sum of the number of search points contained in each of these wedges is equal to SP.) When a wedge contains N search points, each of its sub-wedges can contain, at most,

$N - 1$ search points. (Again, this is an overestimate since the subwedges do not entirely overlap each other.)

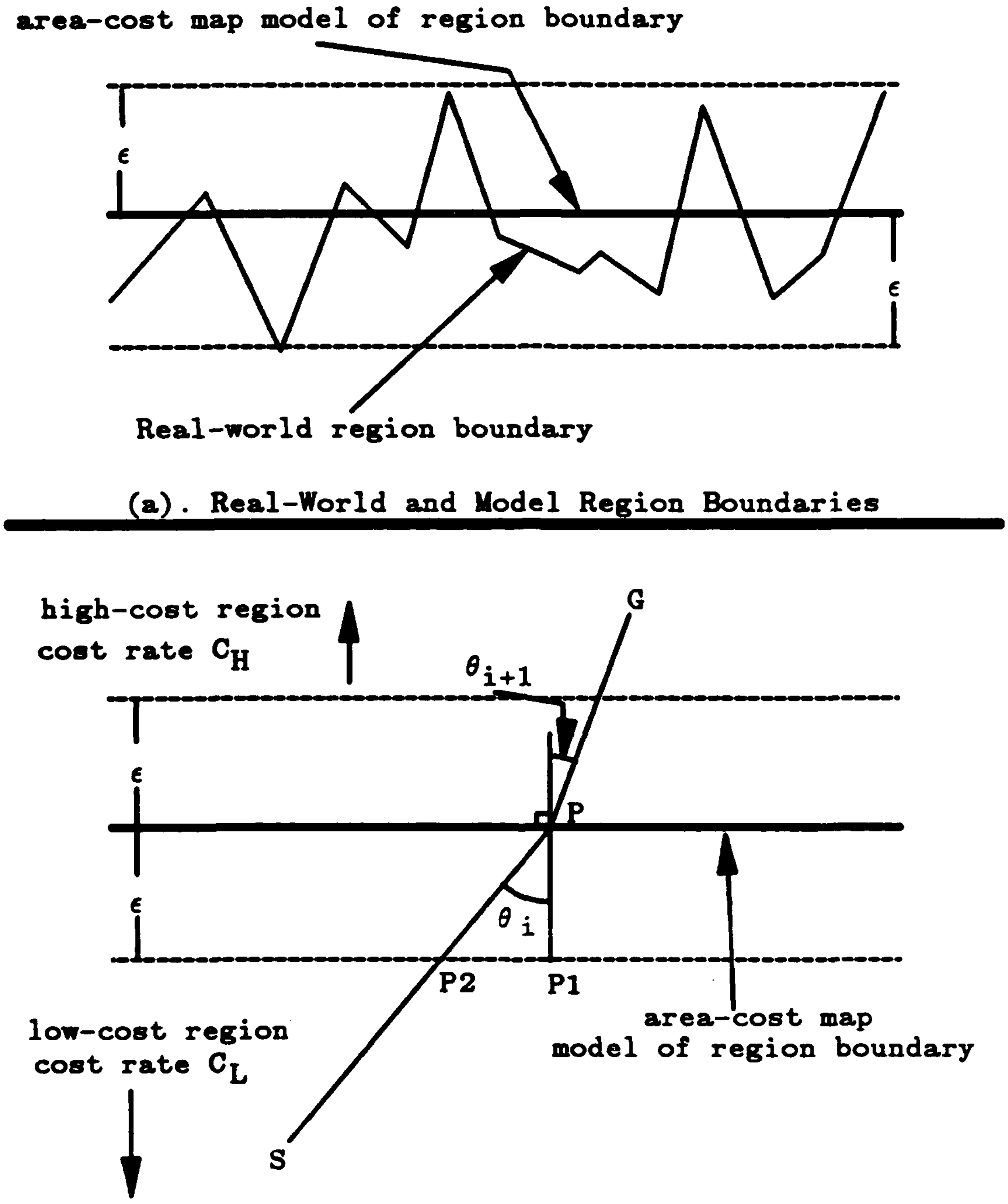
If the initial wedge contains SP search points, in the worst case, that wedge can be refined at most SP times, one refinement based on each search point within the wedge. Due to the upper limit on branching factor, each single refinement can create at most four sub-wedges. As above, each of the sub-wedges created by refinement can contain, at most, $SP - 1$ search points. Thus, each sub-wedge created from the initial wedge can be refined at most $SP - 1$ times. This process can continue until all SP search points are included on the wedge-defining Snell's-law path for every wedge. (Since wedges overlap, the same search point can appear in many different wedges.) There are two initial wedges, so the entire process can be done twice. Thus, $2 \times 4 \times SP!$ wedges can be created in total. Based on this greatest possible number of wedges in the search tree, the algorithm has an exponential worst-case time complexity O($SP!$). Since each wedge created must also be stored, the algorithm also has an exponential space-complexity bound.

We have already noted two ways in which the above analysis for worst-case performance is overly conservative. Also note that the analysis does not consider the pruning of wedges. Further, consider the leaf nodes in a search tree where each wedge has all SP search points on both of the Snell's-law paths that define the wedge. If both of the wedge-defining Snell's-law paths are identical, the wedge would have been pruned after the first refinement. If the wedge-defining paths are not identical then they must intersect in many places (one intersection for each search point on each path). Such intersections terminate wedges which makes further refinement of the wedge impossible. Thus, it should be clear that the exponential time and space bounds are not tight. In Chapter VII, testing results indicate the algorithm's average-case performance is on the order of a quadratic. Specifically, the testing reported in Chapter VII indicates that the average-case performance of the algorithm is on the order of N^2 when there are N search points within the bounding box.

In terms of performance bounds, our Snell's-law-based algorithm seems similar to the well-known simplex algorithm that is widely used in operations research. It has been shown that the simplex algorithm can require an exponential number of steps in the worst case [Ref. 50]. However, its average-case performance is such that it is the most often used algorithm for solving linear programming problems, even though polynomial-time algorithms for the linear programming problem exist [Ref. 51]. The testing results reported in Chapter VII show that our Snell's-law-based algorithm performs well in the average case, despite its exponential worst-case time and space bounds.

A second theoretical measure of our Snell's-law-based algorithm concerns the maximum-cost error in its solution paths. If the problem representation (i.e., the area-cost map) is a perfect representation of the real-world environment and there is no numerical error in the computations, then, based on the derivation of Snell's law in Section IV.C, the solution paths returned by the algorithm have no cost error. However, the problem representation will not, in general, be perfect. If there are large errors in the representation then the cost errors in solution paths will also be large. Our Snell's-law-based algorithm cannot return solution paths that are more accurate than allowed by the problem representation. As an example, if a region vertex is "misplaced" in the area-cost map, the solution path provided by our Snell's-law-based algorithm will rely on the misplaced location for that vertex. Even when the locations of region boundaries on the area-cost map are guaranteed to be within some range ϵ of the actual locations of the corresponding real-world region boundaries, we cannot, in general, bound the error of the Snell's-law solution paths. As an example, if two obstacle boundaries are within ϵ of each other on the area-cost map, they may actually touch in the physical environment. Thus, if the Snell's-law-based algorithm plans a path (based on the area-cost map) between the obstacles, it will not be a feasible path in the real world. We also cannot guarantee that this path could be locally adjusted to become feasible. It could be that the real-world optimal solution path is totally different from the solution path provided by the algorithm.

In some special cases, where multiple region boundaries do not affect the path-cost error, we can bound the cost error of solution paths due to incorrect modeling of region boundary locations. Suppose that each homogeneous-cost region boundary represented on the area-cost map is guaranteed to be within $\pm\epsilon$ of the location of the corresponding real-world boundary. Such an instance might occur when a jagged-edged real-world boundary is modeled by a single line segment of a homogeneous-cost region. If all the "vertices" of the real-world boundary are within 2ϵ of each other, then the resolution of the area-cost map might allow them to be modeled by a single line segment, as illustrated in Figure 124(a). In such a special case, we can bound the cost error of Snell's-law solution paths.



(a). Real-World and Model Region Boundaries

(b). Error Analysis Illustration

Figure 124. Path-Cost Error Due to Misplaced Boundaries

In Figure 124(b), we have illustrated two path segments joined at a single boundary-crossing episode that occurs at a region boundary guaranteed to model the location of a real-world boundary within $\pm\epsilon$. Let the cost rate on one side of the region boundary be C_H and on the other side be C_L and assume that C_H is more expensive than C_L. Suppose that the location of the real-world boundary actually corresponds to the lower dashed line in Figure 124(b). Then, the cost of the S-P path segment has a higher real-world cost than the cost of the Snell's-law solution. In general, the cost difference is:

$$(C_H - C_L)\,\epsilon\,\sec(\theta_i)$$

However, the S-P path segment could have been planned to travel right along the region boundary. In this case, we assume that the path would be executed by traveling along the real-world region boundary and then taking a sharp turn into the high-cost region. Such a path would be S-P2-P1-P-G in Figure 124(b). In this case, the difference in path cost is:

$$C_L\,\epsilon\,\tan(\theta_i) + C_H\,\epsilon - C_L\,\epsilon\,\sec(\theta_i)$$

In some cases, it could occur that the above quantity is greater than the difference in cost rates multiplied by the distance along the P2-P1 path segment. Since we assume that least-costly real-world variation of the planned path would be executed, we also need to consider

$$(C_H - C_L)\,L_i$$

where L_i is the length of the P2-P1 path segment. Let the calculated cost of the S-P (planned) path segment be C_P. When the location of the real-world high-cost region boundary corresponds to the lower dashed line in Figure 124(b), we can bound the error in computed path cost by adding the minimum of the three terms above. Thus, real-world path error along S-P is bounded above by:

$$C_P + \min[(C_H - C_L)L_i, (C_H - C_L)\,\epsilon\,\sec(\theta_i),$$
$$C_L\,\epsilon\,\tan(\theta_i) + C_H\,\epsilon - C_L\,\epsilon\,\sec(\theta_i)]$$

In cases where the location of the real-world boundary corresponds to the upper dashed line of Figure 124(b), the cost of the real-world path is lower than the cost of the computed (Snell's-law) path. In this case, we assume that the path would be executed as planned. Thus, the actual path cost is lower than the computed path cost along path segment P-G of Figure 124(b). The cost difference is:

$$(C_H - C_L)\,\epsilon\,\sec(\theta_{i+1})$$

so that, when C_P is the cost of the P-G segment, the real-world path cost is bounded by:

$$C_P - (C_H - C_L)\,\epsilon\,\sec(\theta_{i+1})$$

Therefore, in special cases where multiple region boundaries do not affect real-world path cost (as described in connection with the obstacle example above), the error involves high-cost region boundaries, and the location of the real-world cost-region boundary is guaranteed to be within $\pm\epsilon$ of their modeled locations, we can establish upper and lower bounds on path-cost error. The total path-cost error is achieved by summing the (local) cost error of each path segment. Note that the angles of incidence and refraction at each boundary-crossing episode are required to compute these errors. Also, the same analysis can be applied to "reversed" situations such as considering the path in Figure 124(b) to be a G-to-P-to-S path.

Numerical computation errors can also affect solution path cost. The algorithm relies on trigonometric functions to complete ray tracing requirements. The numerical routines that compute the trigonometric functions are subject to numerical error. This being the case, we now analyze the amount of cost error in solution paths based on numerical-computation error. Our analysis rests on possible error in computing angular measures, such as angles of incidence, that are used to determine path headings.

In Figure 125, suppose that ϵ is the amount of error in angular measure. If no error were present, a Snell's-law path from S to P1 (on boundary B) would be computed. However, error causes that path to be computed as going from S to P2 on boundary B. We are concerned with the path-cost error that can result from such computation error. In our analysis, we assume that the S-to-P1 path segment is in the same homogeneous-cost region as the S-to-P2 path segment. Thus, the same cost rate can be associated with both path segments. In this case, when we express the length of the error-influenced path segment (h2 for the S-to-P2 path in Figure 125) in terms of the length of the actual (non-error-influenced, as denoted by h1 for the S-to-P1 path segment of Figure 125) path segment we are also expressing the cost of the error-influenced path segment in terms of the cost of the non-error-influenced path segment.

In Figure 125, note that

$$\theta_2 = \theta_1 + \epsilon$$

$$\beta = 90 - \theta_2 = 90 - (theta_1 + \epsilon)$$

$$\alpha = 90 - \theta_1$$

By the law of sines

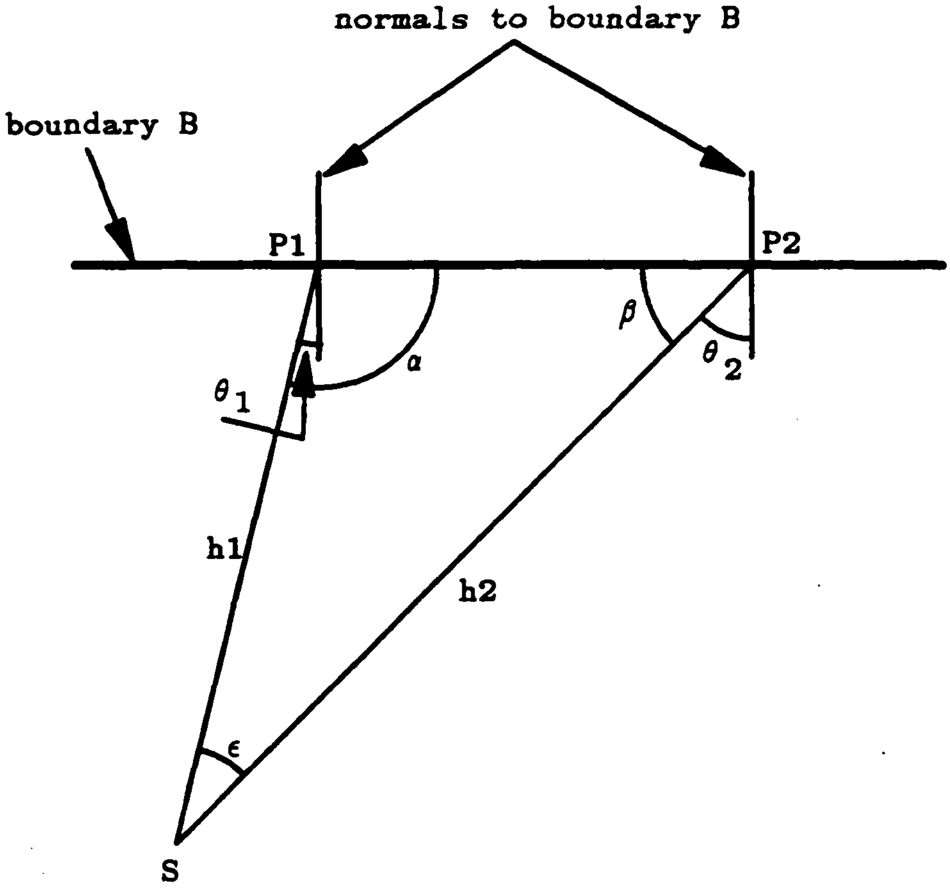


Figure 125. Path Cost Error Illustration

$$\frac{\sin(\beta)}{h1} = \frac{\sin(\alpha)}{h2}$$

$$\frac{\sin(90 - (\theta_1 + \epsilon))}{h1} = \frac{\sin(90 + \theta_1)}{h2}$$

$$\sin(\theta_1 + 90) = \cos(\theta_1)$$

$$\sin(90 - (\theta_1 + \epsilon)) = \cos(\theta_1 + \epsilon)$$

$$\frac{\cos(\theta_1 + \epsilon)}{h1} = \frac{\cos(\theta_1)}{h2}$$

$$h2\cos(\theta_1 + \epsilon) = h1\cos(\theta_1)$$

$$h2 = h1\frac{\cos(\theta_1)}{\cos(\theta_1 + \epsilon)}$$

Now suppose that we can limit the error in angular measure to a maximum of one degree. Then, since $\epsilon = 1$

$$h2 = h1\frac{\cos(\theta_1)}{\cos(\theta_1 + 1)} \tag{2}$$

Allowing θ to range between 0 and 88 degrees, the maximum value of the ratio expressed in equation (2) above is 1.71 and the mean value of the ratio is 1.04 (note that when $\theta = 89$ and $(\theta + 1) = 90$ the ratio is undefined).

This analysis is based on the cost error of a single path segment. However, the same analysis holds (per maximum error) for each path segment in a reported solution path. Thus, each error-influenced path segment could, on the average, have a path segment cost error 1.04 times greater than the cost of the corresponding non-error-influenced path segment, assuming that region boundary headings are randomly distributed. Therefore, when the maximum angular error is limited to one degree, the solution paths reported can have, on the average, a

path cost 1.04 times greater than the cost of the true non-error-influenced solution path. Thus, while the Snell's-law-based algorithm does provide an accurate cost for its solution paths, due to numerical error, these paths may not be the exact minimal-cost solution since they may not obey Snell's law exactly at every boundary-crossing episode.

C. PROLOG IMPLEMENTATIONS

Both the Snell's-law-based algorithm and the wavefront-propagation algorithm are implemented in an interpreted version of C-Prolog that runs under Berkeley UNIX System 4.2. The algorithms were executed on a multi-user Integrated Solutions Optimum V Workstation that has 2MB of main memory and a single 68020 central processor (these machines are similar to the more widely known Sun workstations). The Prolog "cputime" predicate returns the cpu time for a single user; thus the number of users on the machine should not affect timing results. However, only a single user was directly logged onto the machine during test runs.

1. Wavefront Propagation Implementation

The Prolog implementation of the wavefront-propagation algorithm uses many of the ideas developed in Chapter III. Bidirectional search featuring three-way neighbor checking is used. The start- and goal-centered wavefronts are propagated inside the bounding box instead of an ellipse. The slightly larger figure is used so that both the wavefront algorithm and the Snell's-law-based algorithm search the same physical area. The Prolog wavefront implementation does not use a heuristic to selectively expand the wavefronts; the start- and goal-centered wavefronts are uniformly expanded. Thus, with the exception of limiting the physical search space by the bounding box instead of an ellipse, the implemented wavefront algorithm is exactly the Ellipse algorithm of Chapter III (and Tables 10, 11 and 12).

The two-dimensional array is an important data structure for the wavefront-propagation algorithm. Array indices correspond to ternary-cost map coordinates for each node, either directly or by a constant translation. As described in Chapter III, the calculation of neighbor coordinates also allows the random access to the array storing the cost rates for those neighbor nodes. Our version of Prolog does not support the array as a data structure. Instead, lists are available. Lists do not support random access and sequentially searching through a list to find the cost rate for a neighbor cell can be time consuming. We wish to minimize the effect of the lack of the array data structure on the execution time of the wavefront-propagation program. While C-Prolog does not allow random access of lists, it does support direct access to different predicates through a hash table of predicate names. We have used the Prolog "univ" (=..) [Ref. 52,53] operator to construct predicate names that include node X coordinates. As an example, suppose that the lattice node at X coordinate 10 and Y coordinate 20 has a cost rate of 2. For a predicate named "c" (short for cell value) a fact might be "c(10,20,2)". Our scheme instead creates the fact "c10(20.2)", allowing the interpreter to hash into the predicate table based on a node's X coordinate. This scheme markedly reduces execution time of the wavefront program. While the organization is not as ideal as an array, it serves to assure the validity of test results. (We note that the Snell's-law-based implementation is also handicapped by the lack of arrays although this data structure is not as central to the Snell's-law-based algorithm as it is to the wavefront-propagation algorithm.)

There are also several ways in which the lattice-based problem representation could be created. That is, we need a method of assigning cost rates to the lattice nodes, based on the information in the ternary-cost map. There are some difficult questions that must be resolved if the lattice is to accurately reflect the cost-rate information of the ternary-cost map. Since each lattice node represents an area on the cost map, two simple schemes of assigning lattice node cost rates are readily apparent. The average cost rate over the area represented by the node could be assigned or the cost map could be sampled at exactly the map coordinates of the lattice node to obtain a single cost rate. Neither method is perfect. As an example, if the area represented by the node includes some portion of an obstacle area, how does one average in the "infinite" obstacle area cost rate? Similarly, when using a point sampling technique, some important areas of the map might be overlooked. Sampling the area-cost map at exactly the coordinates of the lattice node may "skip over" some important area of the map. Two lattice nodes can be separated on the cost map by a thin obstacle (a fence line for example). Sampling the area-cost map at the coordinates of each node will not capture this situation and can result in an infeasible path (i.e., a path that goes through the fence line) being returned as a solution path.

More powerful techniques could also be used. For instance, a rule-based system might be employed to assign aggregate costs, based on a neighborhood around the node. However. we are not interested. in this work, in improving the wavefront problem representation. The homogeneous-cost region representation seems more appropriate to the problem than any improved version of the lattice-based representation. Thus, we have chosen the simplest lattice-building scheme, that of directly sampling the cost map at the coordinates of the lattice node.

2. Snell's-Law-Based Algori:hm Implementation

There are several Prolog implementations of the Snell's-law-based algorithm that differ only in their lower-bound evaluation functions. The first implementation makes use of cached information about path costs between pairs of region vertices. The cached knowledge is used when developing the initial solution path to construct the bounding box as well as during execution to calculate lower-bound cost evaluations. We denote this version of the algorithm as SL-Dynamic since it uses stored information dynamically, many times during execution. A second version, SL-Static, uses cached knowledge only once, when obtaining an initial solution path. The final version, SL-None, does not use cached knowledge. It obtains an initial solution path by treating the problem as binary and finding the shortest-distance solution path. The cost of this solution path is then calculated on the ternary-cost map.

Other than the different uses of cached information, the implemented Snell's-law-based algorithms are identical and utilize most of the ideas developed in Chapters IV and V. They all rely on a control-flow scheme as depicted in Figure 121. There are some concepts that have been discussed, but not implemented. First, the program does not include a provision to index the search points by their position (as in Section V.D). Each time a portion of a wedge is examined to determine the search points it contains, each search point within the bounding box is considered as a possible candidate for containment. Secondly, precomputed stored optimal-cost paths are not used to physically limit the search space (as in Section IV.I, Figure 81 and Figure 82). We have not addressed the question of how to determine which pair of stored paths best limits the physical space to be searched. As a result, only the bounding box is used to define the area that must contain the optimal-cost solution path. A third unimplemented feature is described in Section V.D. We have not marked portions of wedges which are known not to contain search points (based on the examination of a corresponding wedge portion in a parent wedge). Thus, the Snell's-law-based algorithm must examine the wedge from the wedge tip outward to determine the closest unsolved search point in that wedge. Finally, the implementations do not contain facilities to dynamically alter the size of the bounding box once an initial solution path has been determined (as in Section IV.I). These implementation omissions serve to slow the execution of the Snell's-law-based program but do not invalidate test results.

Recall that the algorithm is designed to solve problems given a ternary-cost map and that this is not a severe limitation (see Section V.A). The implementation only solves problems where each high-cost region and each obstacle region are assumed to be surrounded by optimal-cost area. Again, this is not a severe limitation. It is simply a feature that has not been implemented in the prototype algorithm.

3. Implementation Dependent Characteristics

The version of Prolog that we have used is interpreted; we do not have compiled code for testing. Prolog was chosen for the task because it is a good language for prototyping. However, the execution times reported in this chapter are dramatically different from those cited in Chapter III for the wavefront-propagation algorithms executed in compiled C code. In fact, the mean node-expansion time for the Prolog wavefront-propagation program is over 100 times greater than the mean node-expansion time of the C implementation. While the difference is large, its impact on our comparative analysis is negligible. Both the wavefront program and the Snell's-law-based programs are written in the same language, use the same interpreter, and execute on the same machine. Thus, the timing results are suitable for comparative purposes.

A second consideration involves numerical issues. The Snell's-law-based program is intensive numerically, using several trigonometric functions and relying heavily on line-intersection routines. C-Prolog is not very suitable for programs that have such requirements. The language only supports single precision operations and its trigonometric operators are not entirely accurate. These language-dependent anomalies have necessitated the inclusion of additional predicates to ensure proper operation of the algorithm as the boundary values of numerical routines are approached (arcsine(1.0) as an example). Again, this tends to slow the operation of the Snell's-law-based algorithm.

A final consideration relates to the use of execution time as a comparative performance measure. Some portion of each algorithm's execution time is attributable to the interpreter itself and this time cannot be identified separately. In particular, the Snell's-law-based algorithm makes widespread use of recursion and Prolog's backtracking facility. The wavefront program also relies on backtracking, although to a lesser degree. Backtracking can have a high overhead and the form most often used by the Snell's-law-based program can be more efficiently coded with a standard "if... then... else if... " construct, a facility not available in C-Prolog. Both algorithms also use unifiability [Ref. 52,53] as an "if" construct to affect predicate selection and, again, it is difficult to determine the effect this has on timing marks.

In summary, both algorithms can be more efficiently coded in other languages. Prolog has been chosen because it is very supportive of prototyping requirements; code is produced easily and quickly. We have taken steps to lessen the impact of Prolog's shortcomings for both algorithms. However, one should be aware of the Prolog characteristics that make the language somewhat ill-suited to the implementations of the path-planning algorithms. Still, both algorithms execute on the same hardware/software system and thus the timing marks are indicative of their relative performance.

D. SELECTING TEST PROBLEMS

Two ternary-cost maps are used in the testing process. The first map, Map1, is entirely artificial and was designed to exhibit a variety of homogeneous-cost region geometry. A large amount of stored path cost information has been accumulated for this map. (Thus, problems solved by the SL-Dynamic and SL-Static algorithms come from Map1.) Specifically, we have stored information that can be used to directly construct the optimal-cost path between any two region vertices on the map when a 2:1 cost ratio exists between the high- and low-cost traversable areas. Recall (Section IV.H) that entire paths between region vertices need not be stored for this purpose. Only those links that do not include region vertices as intermediate turn points are required. Using this scheme, 301 bidirectional links are required to represent the optimal-cost paths between the 44 region vertices of Map1; a map that has 4 obstacle and 4 high-cost regions. Figure 126 depicts Map1 as well as reference grid where reference lines appear eight units apart. (The units are generic; they are the map coordinate units.) The reference lines are not the lattice used by wavefront propagation. The reference grid is simply included so that some of the test problem start and goal coordinates can be positioned in Figure 126, if desired. The lower left corner of the grid has coordinates (0,0). In the figure, the dark polygons represent obstacle areas, the light polygons depict high-cost traversable regions, and the unshaded background area is the low-cost traversable area.

The second ternary-cost map, Map2, represents a portion of the Point Lobos ternary-cost map introduced in Chapter III. Only a portion of the Chapter III ternary-cost map is used since we wish to limit the size of the problem representation for both maps without losing detail. Also, to make Map2 more interesting, some obstacle areas from the Chapter III version have been moved slightly so that they are included on the ternary-cost map. Map2 has 85 region vertices to describe its two obstacle areas and its 4 high-cost areas. Figure 127 shows a reference grid superimposed onto Map2. Again, there are eight units between reference lines, the lower left corner of the map is located at the origin of the coordinate system and the same shading of regions is used.

Both ternary-cost maps represent an area limited by 128 square units. The Snell's-law-based algorithm is able to use rational-numbered values for path turn points and for the coordinates of region vertices. To maintain consistency with the wavefront program, only integer values have been used as region vertex

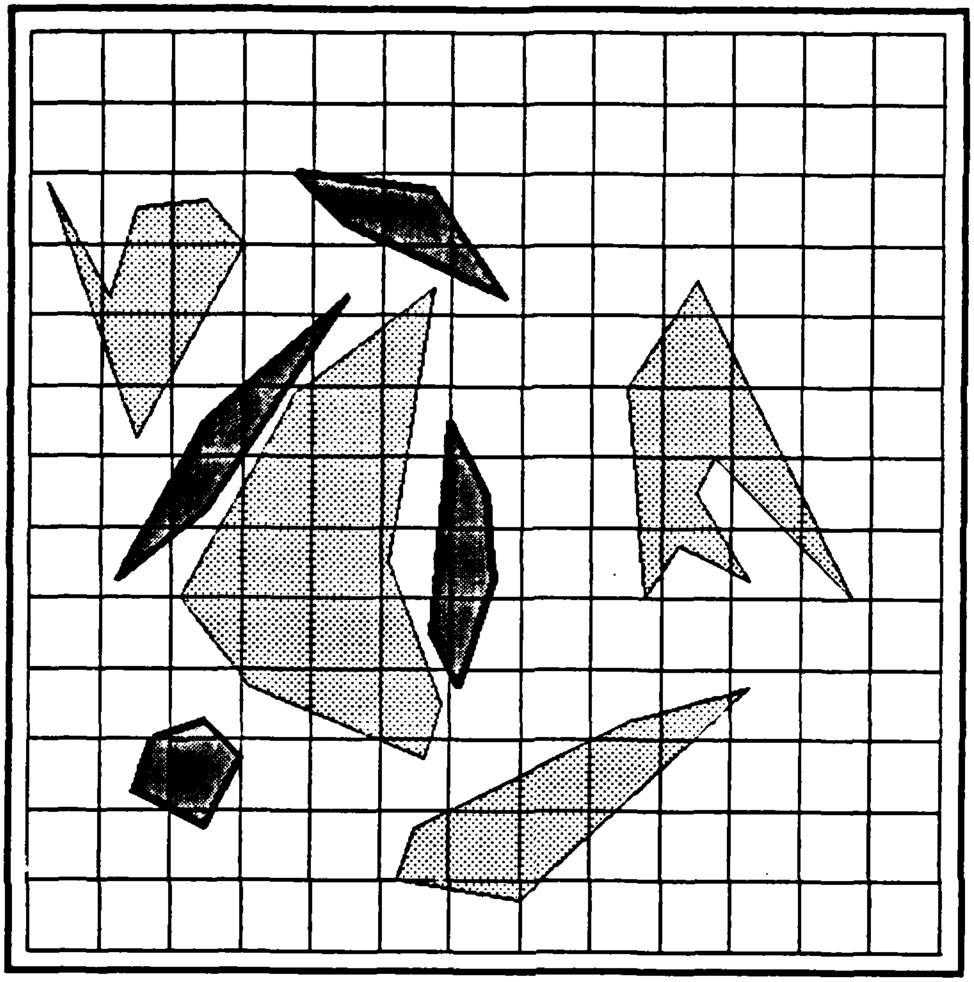


Figure 126. The Map1 Area-Cost Map

coordinates. (Thus, the Pt. Lobos map used here is a "rounded" version of that used in Chapter III.) Also, this wavefront implementation does not rely on the screen pixel (as in Chapter III) as the highest unit of resolution. Rather, an independent unit is used so that each ternary-cost map can be entirely represented by a 128 by 128 node lattice. When this number of lattice nodes is used to represent the ternary-cost map, we refer to the resolution as 1:1. The wavefront program can be executed at different resolutions that are integer multiples of the basic 1:1 resolution. When the resolution is lowered by a factor of

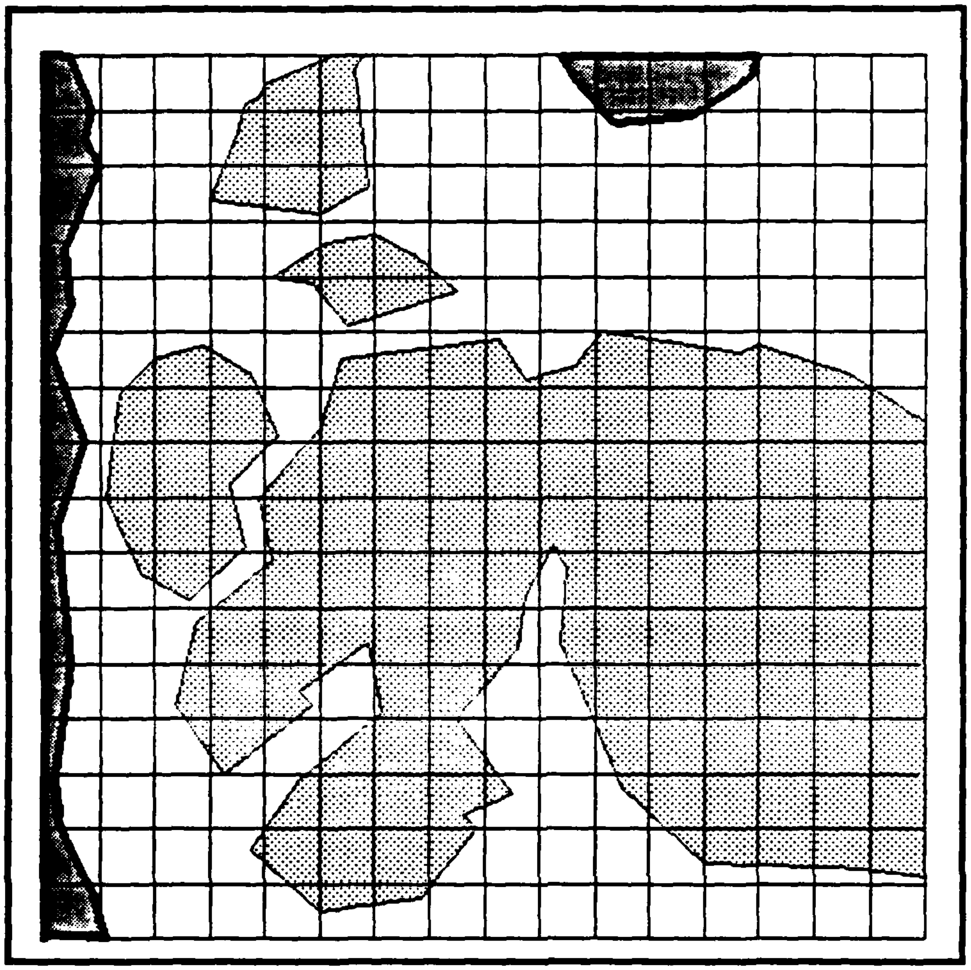


Figure 127. The Map2 Area-Cost Map

two, the same ternary-cost map is fully described by a lattice of 64 nodes by 64 nodes and we refer to this scheme as having a 2:1 resolution (2 square units are represented by 1 lattice node). The next section includes some results for wavefront algorithms solving identical problems at different resolutions.

The different algorithms have been executed on a variety of test problems. Some of the problems have been chosen to illustrate particular algorithm behavior. As an example, a problem that features a bounding box having a small area is favorable to the wavefront algorithm (in terms of the time required to solve the problem). Note that the area of the bounding box approximately corresponds to the number of nodes in the lattice at a 1:1 resolution. Thus, when the lattice is small, the low overhead of the wavefront technique should allow that algorithm to quickly solve the problem. Similarly, a problem featuring a large bounding box that includes only a few search points should be favorable to the Snell's-law-based algorithm since this technique is insensitive to the bounding-box area but is sensitive to the number of region vertices that must be considered. The start and goal locations for some problems have also been chosen at random.

For each wavefront problem, we have recorded the area of the bounding box, the number of lattice nodes expanded and the time required to solve the problem. The area of the bounding box is on the order of the storage space required to represent the problem to the wavefront algorithm. The Snell's-law-based technique has been used to solve an identical set of problems. For it, we have recorded the number of search points in the bounding box, the number of wedges made, the number of wedges searched, the number of line intersections calculated, and the time required to solve the problem. The number of search points is on the order of the storage requirement to represent the problem. The number of intersections required during the algorithm's execution provides a general characterization of the time requirement of the Snell's-law-based algorithm, much as the number of nodes expanded describes the wavefront technique. That is, knowing the time required to find the intersection of two lines on another machine in another language can be used to make a reasonable prediction of the Snell's-law-based algorithm's time requirements on that machine in that language.

E. COMPARATIVE DATA

In this section, we first present the time and space performance of the SL-Dynamic algorithm (Snell's-law-based using cached information when finding an initial solution path a well as dynamically to achieve lower-bound cost evaluations) and the 1:1 resolution wavefront-propagation algorithm. The test problems are based on Map1 and a 2:1 cost ratio between high- and low-cost traversable areas. In Table 14, we give a one or two letter code to identify test problems (i.e., start and goal points). The coordinates of the start and goal can be used to approximately position any problem on the maps of Figures 126 and 127.

In Table 15, we show some test results. We list the problem key ("Prob #"), the bounding box area in coordinate system units ("Box Area"), the number of search points in the problem ("SP"), the number of nodes expanded by the wavefront algorithm ("Nodes Exp"), the time required to achieve a wavefront solution path ("Time to Solve"), the number of line intersections calculated (both attempted and successful) by the SL-Dynamic algorithm ("# of Ints"), and the time required for the SL-Dynamic algorithm to reach a solution path ("Time to Solve"). Recall that the number of search points (and thus the problem description space) includes original region vertices, artificial region vertices (i.e., boundary points), and the goal. Table 15 spans two pages.

Problems a and k exemplify cases where a very small bounding box area allows the wavefront algorithm to solve the problem in less time than the Snell's-law-based algorithm required. (Note that the wavefront method still requires more problem-description space based on a comparison of bounding-box area versus the

TABLE 14

PROBLEM DESIGNATION MAP1 PROBLEMS			PROBLEM DESIGNATION MAP1 PROBLEMS		
Prob #	Start (X,Y)	Goal (X,Y)	Prob #	Start (X,Y)	Goal (X,Y)
a	(78,23)	(75,31)	ac	(52,16)	62,22)
b	(78,16)	(71,33)	ad	(31,40)	(34,54)
c	(81,51)	(44,54)	ae	(31,40)	(34,46)
d	(77,53)	(71,77)	af	(52,16)	(78,28)
e	(72,38)	(92,58)	ag	(90,10)	(4,92)
f	(49,1)	(57,23)	ah	(75,44)	(77,53)
g	(51,10)	(13,70)	ai	(8,78)	(14,59)
h	(8,56)	(23,84)	aj	(78,45)	(77,58)
i	(19,31)	(43,68)	ak	(14,86)	(13,39)
j	(19,31)	(38,70)	al	(8,61)	(48,89)
k	(29,19)	(15,27)	am	(88,48)	(43,9)
l	(13,35)	(31,72)	an	(88,48)	(75,27)
m	(13,7)	(45,100)	ao	(22,78)	(13,83)
n	(56,38)	(8,78)	ap	(78,4)	(80,38)
o	(64,48)	(92,51)	aq	(43,25)	(40,92)
p	(24,54)	(7,81)	ar	(38,23)	(46,33)
q	(72,19)	(77,58)	as	(68,41)	(80,71)
r	(43,48)	(26,58)	at	(69,41)	(80,70)
s	(11,72)	(40,91)	au	(49,59)	(44,74)
t	(48,4)	(8,28)	av	(18,13)	(11,45)
u	(60,18)	(40,69)	aw	(48,2)	(46,16)
v	(60,18)	(79,61)	ax	(31,2)	(43,41)
w	(29,10)	(58,50)	ay	(71,4)	(64,18)
x	(7,75)	(40,91)	az	(72,11)	(80,51)
y	(8,17)	(40,30)	ba	(71,12)	(80,51)
z	(60,31)	(47,58)	bb	(88,32)	(96,128)
aa	(42.66)	(51.82)	bc	(1.81)	(25.128)
ab	(52,16)	(66,24)			

number of search points within the bounding box.) Problems bb and bc are favorable to the Snell's-law-based method, in that the associated bounding boxes encompass a large area but only a few search points. Problems i and j exemplify the large performance difference that can be based on the accuracy of the initial solution path (used to construct the bounding box and thus limit the search space). In problem i, the goal is inside the large high-cost region near the center

TABLE 15
SL-DYNAMIC. 1:1 RESOLUTION WAVEFRONT COMPARISON

Prob	Box	SP	Wavefront Algorithm		SL-Dynamic Algorithm	
#	Area		Nodes Exp	Time to Solve (sec)	# of Ints	Time to Solve (sec)
a	90	5	87	5.95	79	9.28
b	304	6	297	23.53	304	10.88
c	1532	21	1282	123.33	675	67.36
d	1211	12	1062	108.67	232	33.28
e	759	12	689	60.23	340	36.11
f	701	9	543	49.58	159	15.45
g	6989	42	5115	692.1	3237	265.47
h	374	6	317	26.98	113	12.8
i	5993	42	3462	483.42	797	74.75
j	744	13	551	56.73	175	14.58
k	65	5	44	2.96	37	3.51
l	1497	16	1084	109.80	272	18.76
m	3690	28	3030	411.76	1190	95.75
n	4345	39	3527	423.17	1271	96.46
o	1246	9	1149	106.17	373	33.08
p	3207	24	2393	290.02	662	53.95
q	1086	14	970	108.05	299	39.02
r	1157	17	624	68.81	318	28.91
s	1847	20	1419	132.45	637	50.56
t	507	9	407	27.85	68	8.67
u	2442	31	1964	243.73	964	99.01
v	3061	24	2544	319.70	1226	129.13
w	617	11	446	35.07	167	12.87
x	1054	14	761	64.53	344	34.15
y	1411	11	1137	98.75	252	19.35
z	515	8	416	33.32	96	7.01
aa	501	14	384	29.61	133	17.20
ab	900	15	355	32.02	100	12.36
ac	471	10	229	14.35	30	3.23
ad	710	12	338	37.48	119	9.28
ae	156	1	74	5.76	7	0.73
af	2840	23	1368	132.32	307	36.20
ag	4814	35	4077	456.53	1285	117.51
ah	179	11	146	11.24	107	11.22

of the ternary-cost map. The initial solution path includes a long high-cost segment from the goal to the high-cost region boundary closest to the start (at the southern end of the high-cost region). In problem j, the goal is outside the same

TABLE 15 (continued)
SL-DYNAMIC, 1:1 RESOLUTION WAVEFRONT COMPARISON

Prob	Box	SP	Wavefront Algorithm		SL-Dynamic Algorithm	
#	Area		Nodes Exp	Time to Solve (sec)	# of Ints	Time to Solve (sec)
ai	959	16	709	62.55	261	30.91
aj	238	8	152	11.87	134	16.76
ak	3253	20	2529	314.02	655	57.31
al	1817	22	1318	119.10	578	51.91
am	5436	31	3070	389.97	912	72.73
an	1833	13	965	100.30	316	37.47
ao	367	10	312	23.53	243	21.85
ap	201	4	199	18.18	44	5.27
aq	3543	32	2773	391.80	1535	135.88
ar	247	9	196	15.70	82	9.63
as	1297	13	1216	126.41	311	33.68
at	1434	16	1344	141.31	338	36.48
au	178	9	165	12.6	71	8.51
av	438	13	328	26.43	160	13.05
aw	186	5	160	12.48	88	8.45
ax	769	8	642	65.05	108	10.05
ay	213	3	180	12.7	26	2.88
az	537	11	510	50.46	133	12.18
ba	581	12	518	51.04	148	14.53
bb	1776	7	1562	234.91	92	8.96
bc	312	4	310	28.6	65	6.70

high-cost region, just above and to the left of the problem i goal. The initial solution path for this problem consists of only optimal-cost links that travel around the high-cost regions, and results in a much smaller bounding box. Clearly, a more intelligent initial-solution path would speed up performance for problem i.

The raw data of Table 15 show that the SL-Dynamic algorithm required less problem description space for every test problem. This is not an absolute guarantee however. In cases where region vertices are so numerous that there is nearly a 1:1 correspondence between them and the wavefront lattice nodes, the

Snell's-law-based method can require more storage space due to the inclusion of artificial boundary points (which, in general, have rational-valued, not integer coordinates). In most problems, the space required to store homogeneous-cost region problem representations is much less than that required for lattice-based representations that capture the same level of detail.

In Table 16, we summarize the mean node-expansion time (for a single node) and the mean intersection time (for a single intersection) for the respective algorithms. These measures provide reasonable characterizations of the two methods that can be used to predict their approximate performance on other host hardware/software systems. Recall from Chapter III that the Ellipse wavefront-propagation method had a mean node-expansion time of 0.0008 seconds. The vast difference between the compiled C-coded wavefront algorithm and the Prolog version provides a general measure of possible performance improvements. While the difference is not an absolute standard, it is indicative of the performance improvement that can be made to the Snell's-law-based algorithm by recoding in another more efficient language.

TABLE 16
MEAN PERFORMANCE FOR ALL TEST PROBLEMS (in seconds)

Wavefront		SL-Dynamic	
Mean Node Expansion	Standard Deviation	Mean Intersection	Standard Deviation
0.1035	0.0276	0.0997	0.0161

Table 15 also evinces the time-requirement superiority of the Snell's-law-based algorithm. Only in small problems is the time required by the SL-Dynamic algorithm greater than that required by the wavefront technique. The fact that the wavefront algorithm solves small problems quickly suggests that decreasing its resolution can allow the algorithm to achieve better time and space performance than the Snell's-law-based method. In Table 17, we present test results that make this issue more clear. Here, we have used some of the same test problems listed in Table 15. Table 17 reflects the performance of a 2:1 resolution wavefront algorithm. Note that changing the resolution by a factor of 2 decreases the bounding box area by a factor of 4. Also, we are still relying on point sampling to construct the problem-representation lattice.

TABLE 17
ALTERING RESOLUTION

Prob	1:1 Wavefront		2:1 Wavefront		SL-Dynamic	
#	Area	Time (sec)	Area	Time (sec)	SP	Time (sec)
e	759	60.23	190	15.36	12	36.11
f	701	49.58	175	14.3	9	15.45
n	4345	423.17	1086	74.22	39	96.46
af	2840	132.32	710	30.13	23	36.20
ag	4814	456.53	1204	94.93	35	117.51
ai	959	62.55	240	16.25	16	30.91
al	1817	119.10	455	29.05	22	51.91
aq	3543	391.80	886	55.49	26	135.88
as	1297	126.41	325	23.95	13	33.68
az	537	50.46	135	13.8	11	12.18
bb	1776	234.91	444	30.32	7	8.96

Some testing has also been completed for Map2 problems. This map has a greater number of region vertices and a larger portion of the map is in high-cost areas. The Map2 problems produce the same type of result as those that come from Map1. In Table 18, we define the (key) "Prob #" designations for a sample of these problems. Table 19 presents the results of using a 1:1 resolution wavefront, a 2:1 resolution wavefront, and the SL-None algorithms to solve the same set of problems.

TABLE 18

PROBLEM DESIGNATION MAP2 PROBLEMS			PROBLEM DESIGNATION MAP2 PROBLEMS		
Prob \#	Start (X,Y)	Goal (X,Y)	Prob \#	Start (X,Y)	Goal (X,Y)
A	(7,12)	(72,47)	F	(38,9)	(92,20)
B	(44,37)	(72,91)	G	(10,12)	(64,26)
C	(12,22)	(44,104)	H	(35,70)	(7,78)
D	(67,29)	(37,111)	I	(97,96)	(27,108)
E	(12,22)	(36,80)	J	(20,80)	(100,104)

TABLE 19
ALTERING RESOLUTION (Map2 Problems)

Prob	1:1 Wavefront		2:1 Wavefront		SL-None	
\#	Area	Time (sec)	Area	Time (sec)	SP	Time (sec)
A	6174	682.35	1544	85.22	39	201.08
B	9380	1529.13	2345	145.12	64	569.58
C	8307	1244.65	2076	214.18	56	346.26
D	20418	3047.22	5105	973.05	83	1212.13
E	2986	308.88	747	57.37	62	74.20
F	4928	379.73	1232	87.37	26	117.08
G	5185	531.63	1297	103.76	30	197.88
H	2148	176.45	537	35.61	34	79.35
I	3846	346.75	962	68.03	34	120.63
J	3997	388.65	1000	77.25	34	72.58

Tables 17 and 19 clearly show that, for small problems, the low overhead of the wavefront-propagation algorithm allows it to find a solution path in less time than that required by the Snell's-law-based algorithm. particularly when there is a large number of search points inside the bounding box. We discuss this result in more detail in Chapter VII. A pertinent question that arises here however, involves the cost of gaining improved time efficiency by decreasing resolution. The amount of error, as measured by solution path cost, is essentially random.

F. WAVEFRONT PROPAGATION PATH-COST ACCURACY

We have seen that decreasing the resolution of the wavefront propagation lattice decreases the time requirements of the algorithm. We now demonstrate how decreasing representation resolution can have a random effect on the accuracy of resulting solution paths. In Figures 128 through 135, we show only the portion of the ternary-cost map that remains inside the bounding box (after the initial solution path has been found). Each figure also shows the bounding box itself. The solidly-darkened polygons in the figures represent obstacles while the lightly-shaded polygons depict high-cost traversable areas. The unshaded background is the optimal-cost traversable area. The series of thick line segments connecting the start and goal is the Snell's-law-based algorithm solution path. The series of thinner line segments between those two points is a wavefront solution path (at the resolution indicated in the figure, 1:1 is high resolution, 2:1 is low resolution).

In each problem illustrated in Figures 128 through 135, the Snell's-law algorithm provides the lowest-cost solution path. This phenomena occurred in every test problem reported in this chapter; the Snell's-law-based method always provided a lower-cost solution path than the wavefront-propagation algorithm. Based on this evidence and the fact that Snell's law results as a consequence of using derivatives to characterize minimum-cost paths (as developed in Section IV.C), we hypothesize that the cost of a Snell's-law-based solution path is a lower bound on the cost of a wavefront-propagation path (for the same problem). We have not been able to prove this hypothesis due to the effects of numerical-computation errors on the Snell's-law solution paths. However, we know that the

Snell's-law method provides lower-cost solution paths in each of the example problems below. Thus, in the following discussion, we assume that the Snell's-law solution is the standard and that the wavefront algorithm solution paths have some *% added cost* relative to the cost of the Snell's-law solution path.

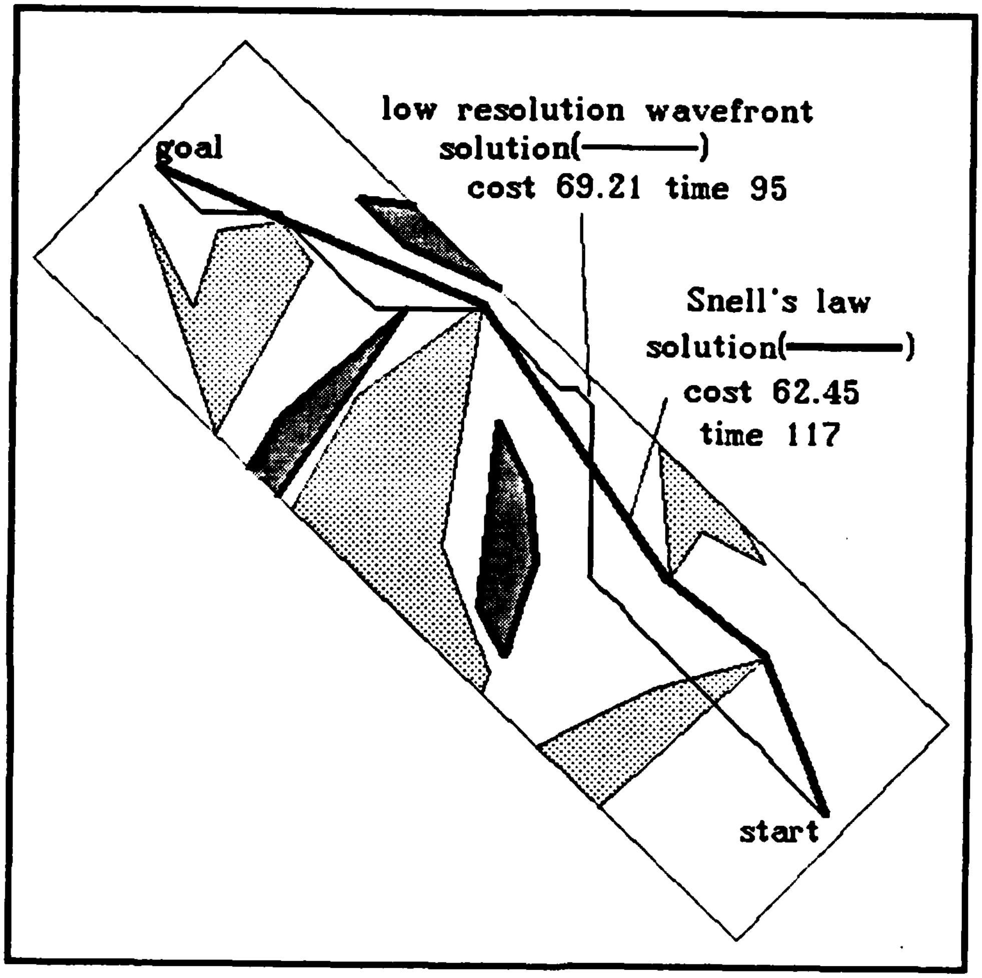


Figure 128. Problem ag, Low Resolution

Figures 128 and 129 display the low- and high-resolution wavefronts solution paths to the same problem compared to the Snell's-law-based solution path. This is problem ag of Table 17. The high-resolution wavefront solution path has an added cost of 7.8% when compared to the Snell's-law-based solution path. The

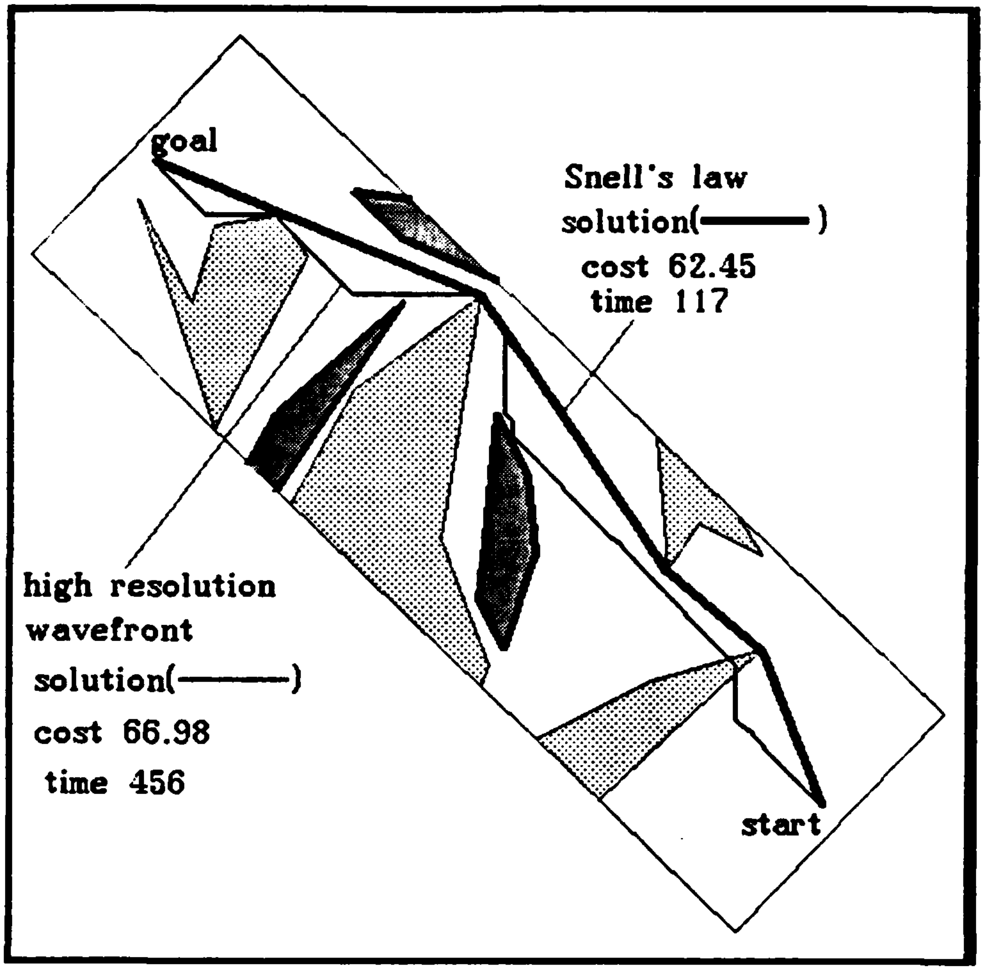


Figure 129. Problem ag, High Resolution

faster low-resolution algorithm returned a solution path having 10.8% added cost. Figures 130 and 131 show the solution paths achieved by the three methods for problem al. Again, the high-resolution wavefront solution path is a better approximation to the Snell's-law-based solution path. having an added cost of 8.1% while the low-resolution wavefront solution path has a 10.35% added cost. Note that both wavefront algorithms provide solution paths that go through an obstacle region (near the goal). This is due to a modeling anomaly in our point-sampling method of constructing the lattice. The resolution of the lattice is not high enough in either case to capture the fact that some of the solution path is inside an obstacle area. These solution paths suggest that even a 1:1 resolution is not sufficient to ensure strict solution path feasibility.

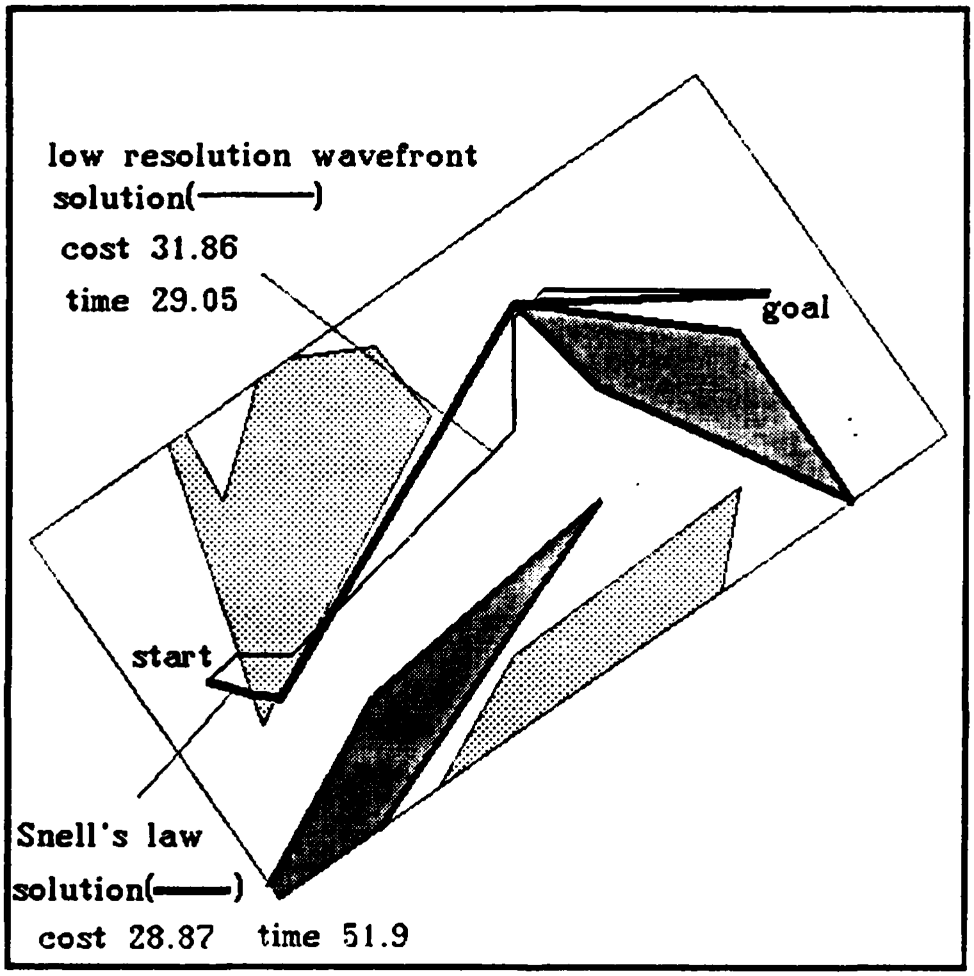


Figure 130. Problem al, Low Resolution

Figures 132 and 133 both depict solution paths to the Map2 problem G of Table 19. Note that both wavefront algorithms yield solution paths that are very different physically from the Snell's-law-based solution path. The non-digitally-biased version of the path represented by the wavefront solution paths was actually found by the Snell's-law-based algorithm during execution. However, the

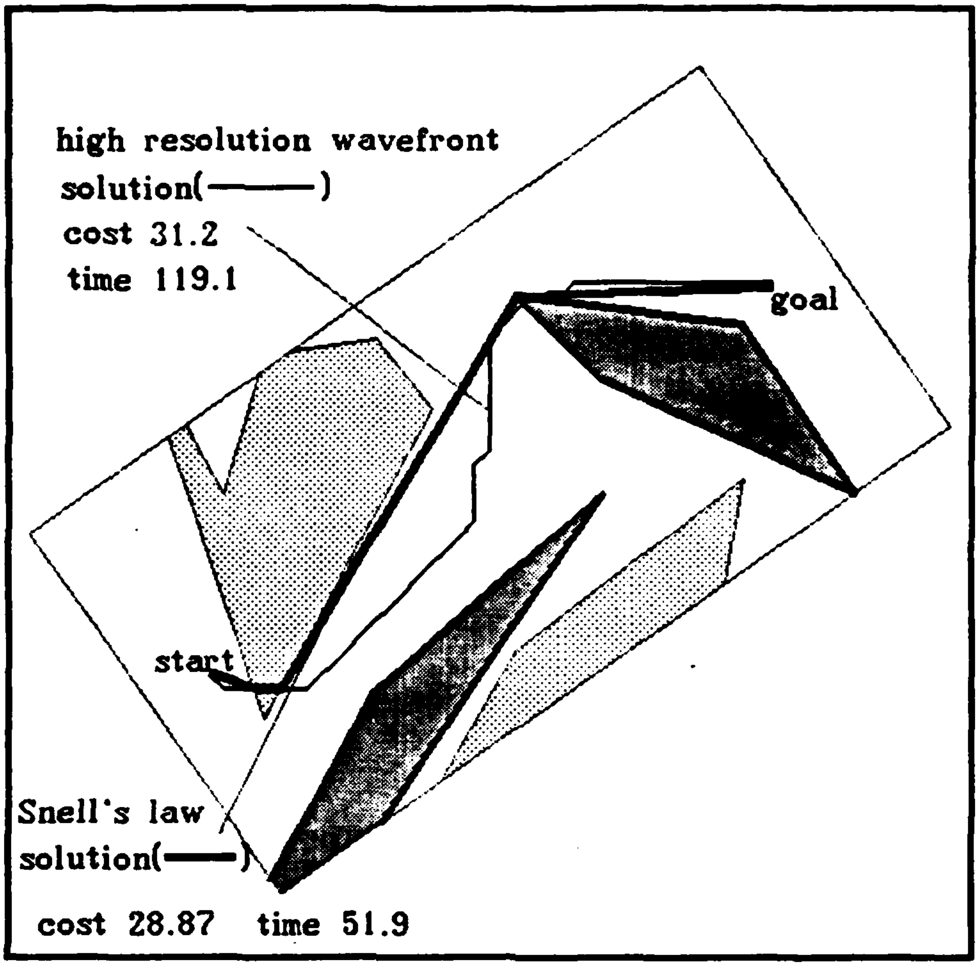


Figure 131. Problem al, High Resolution

path has a slightly higher cost (37.21) than the "southern" route selected as the optimal-cost solution path. Both wavefront solution paths have approximately 7% added cost.

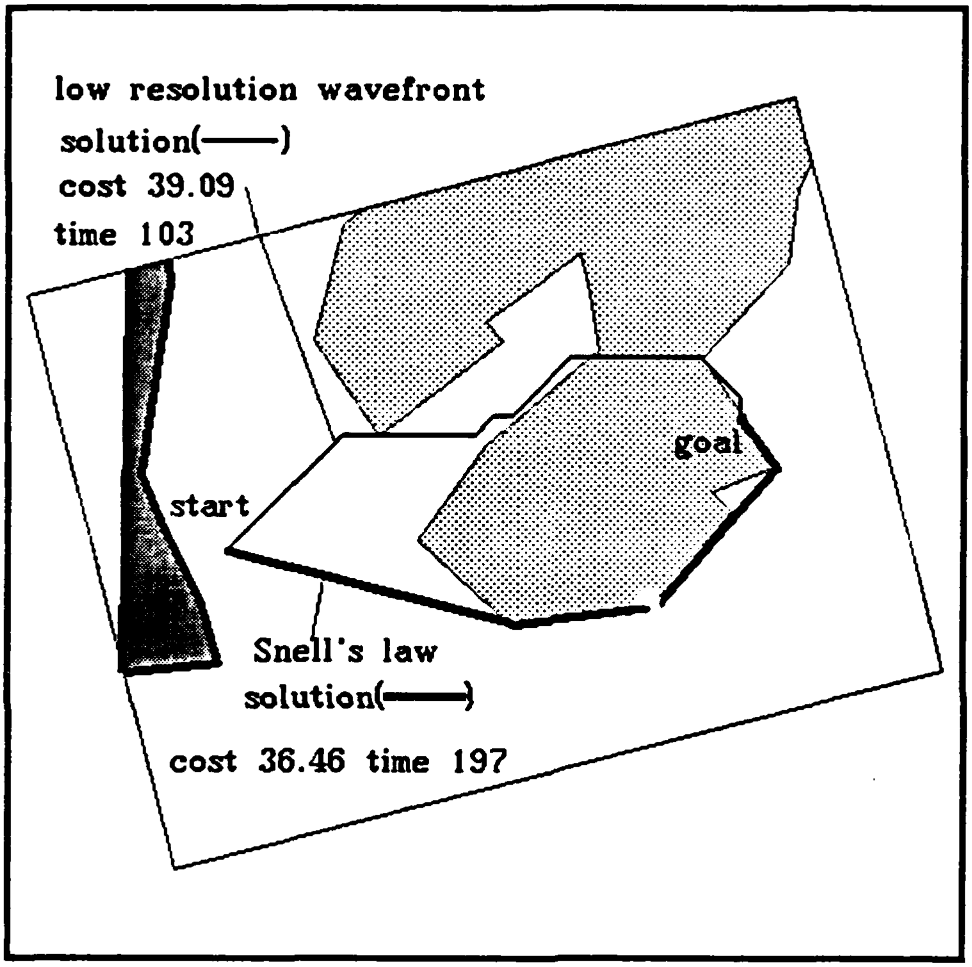


Figure 132. Problem G, Low Resolution

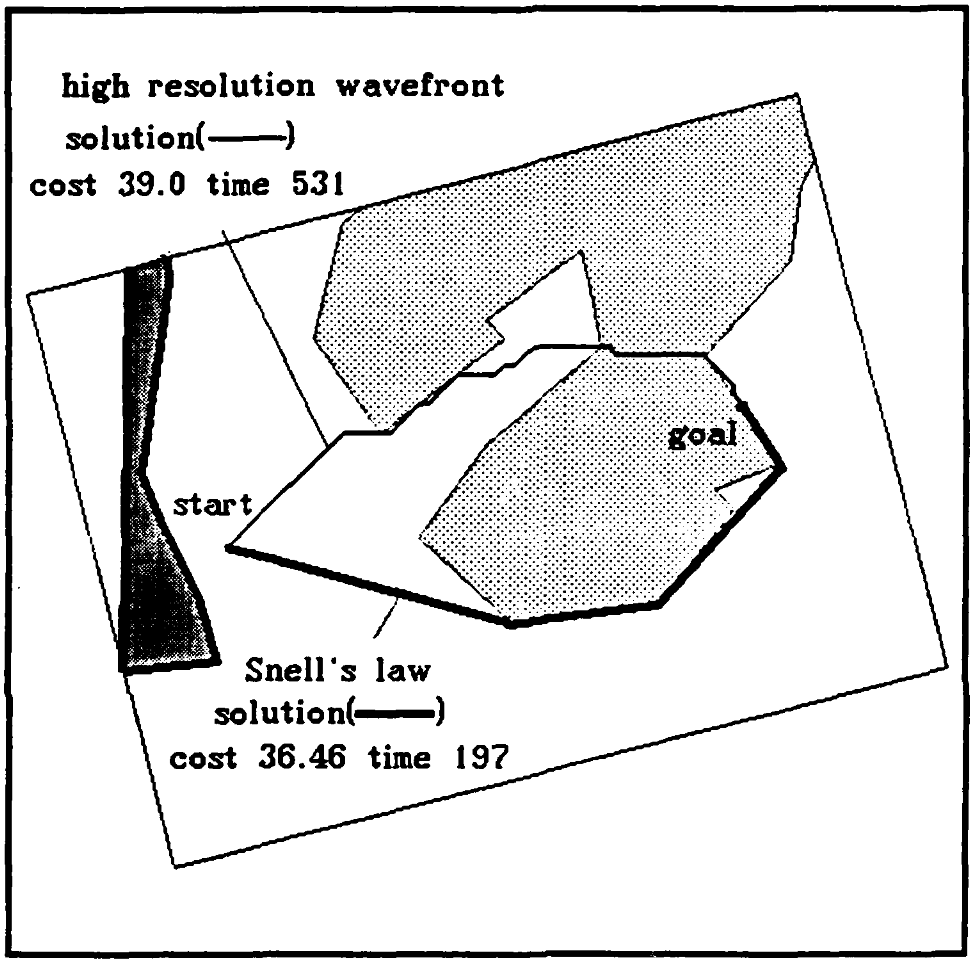


Figure 133. Problem G, High Resolution

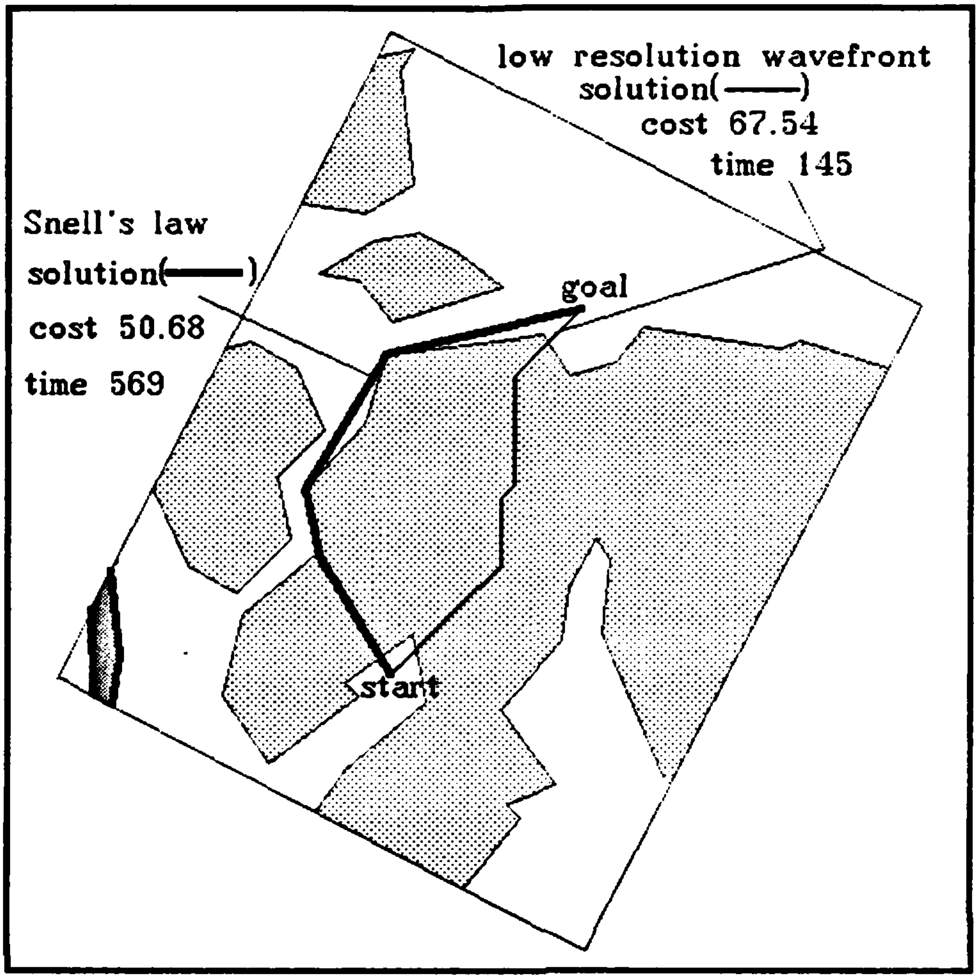


Figure 134. Problem B, Low Resolution

Figures 134 and 135 clearly exhibit the random nature of the % added cost caused by low-resolution problem representations. (These figures depict solution paths to problem B of Table 22.) The 2:1 wavefront algorithm misses the optimal-cost "alley" included in the solution path found by the other two methods. This results in the 33.3% added cost of the low-resolution wavefront solution path. The high-resolution solution path has a 7.9% added cost.

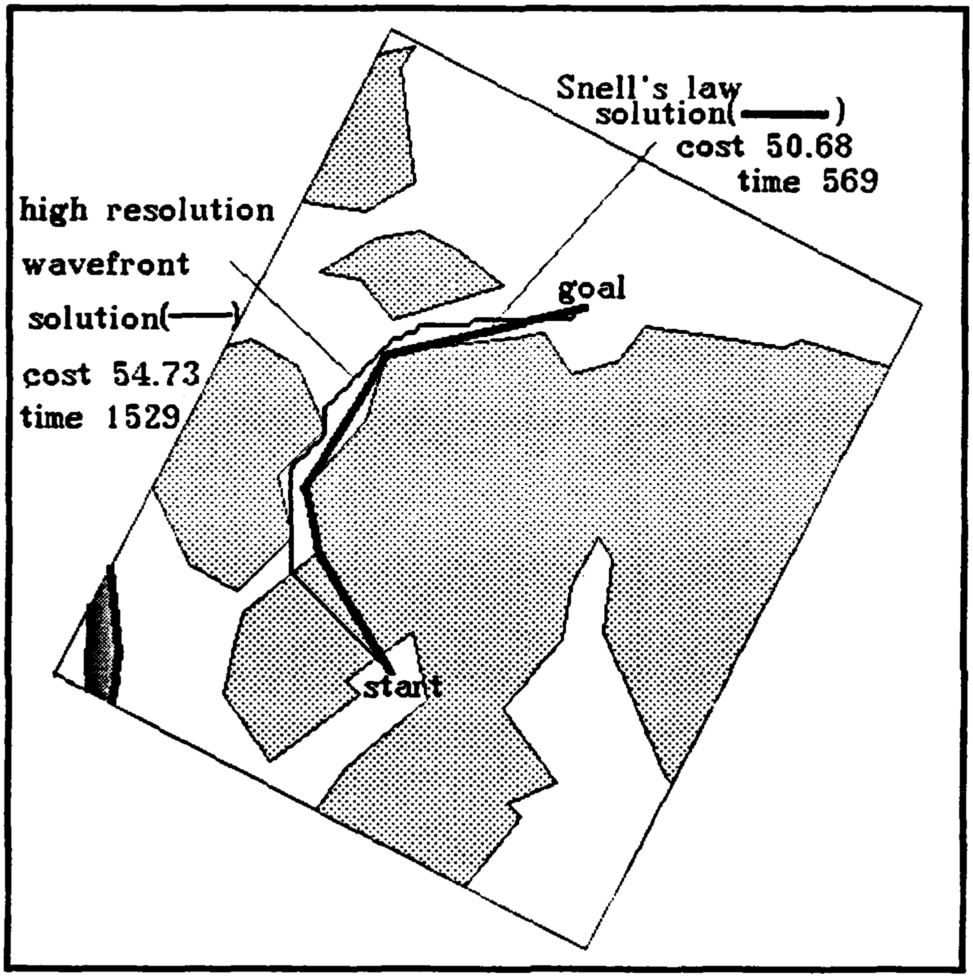


Figure 135. Problem B, High Resolution

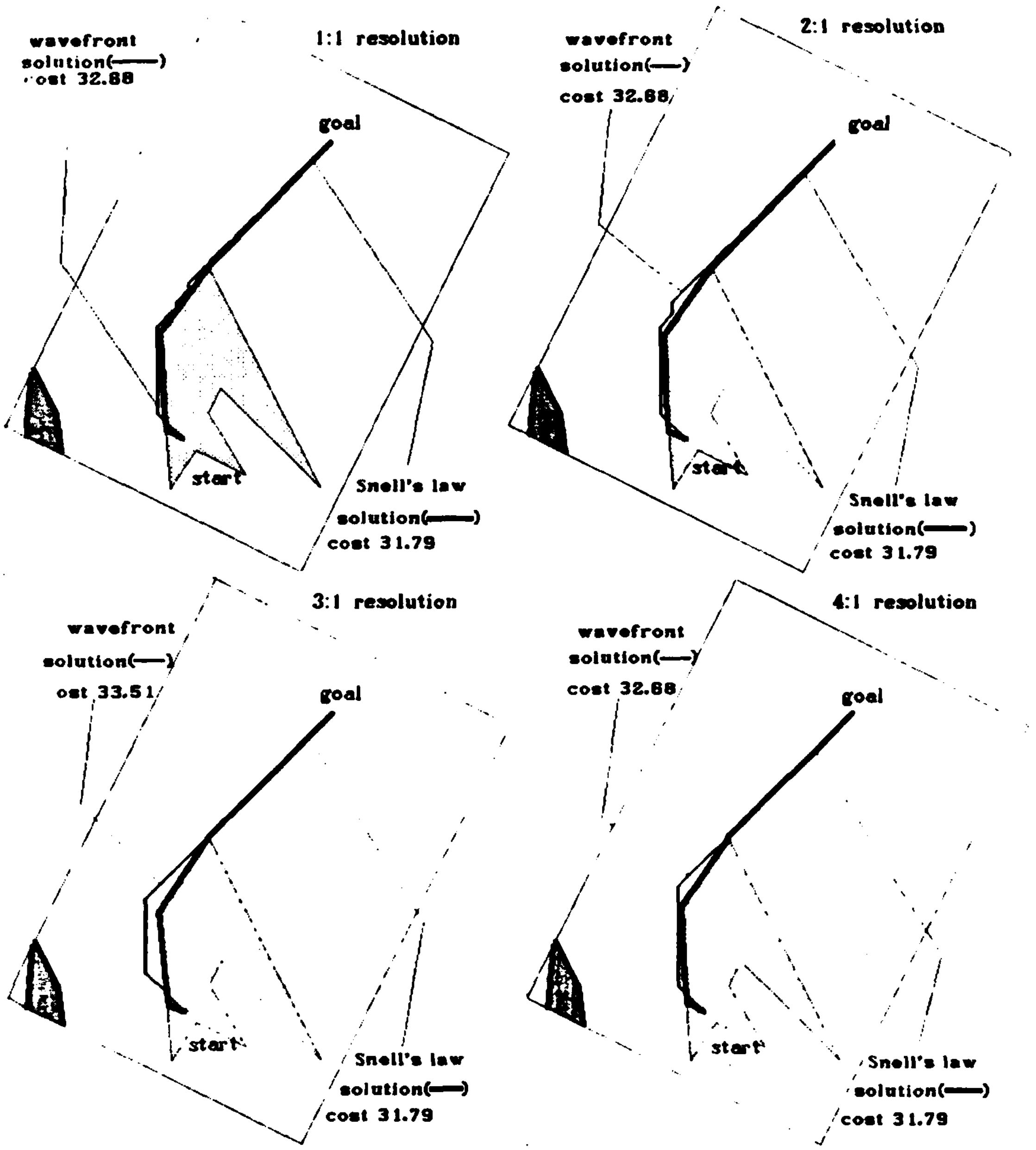


Figure 136. Problem 1, 1:1, 2:1. 3:1 and 4:1 Resolutions

To more clearly exhibit the random nature of % added cost due to resolution, Figures 136 through 139 depict the wavefront solution paths to two problems, all computed at resolutions of 1:1. 2:1, 3:1, 4:1. 6:1. 8:1, 12:1 and 24:1. Figures 136

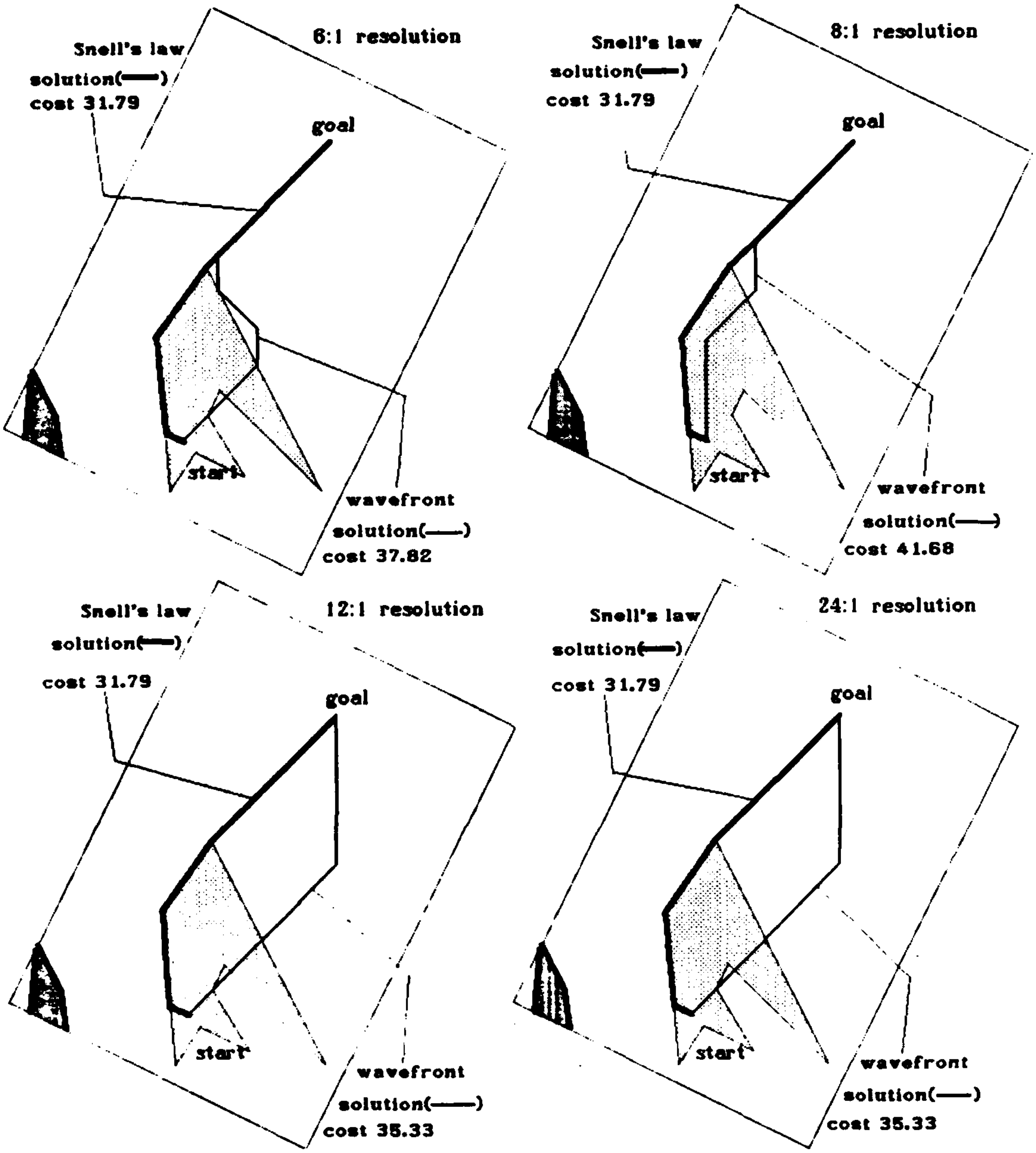


Figure 137. Problem 1, 6:1, 8:1, 12:1 and 24:1 Resolutions

and 137 show solution paths to a Map1 problem while Figures 138 and 139 come from a Map2 problem. Note that in both problems, cases occur where a lower-resolution problem representation allows a more accurate solution path than that

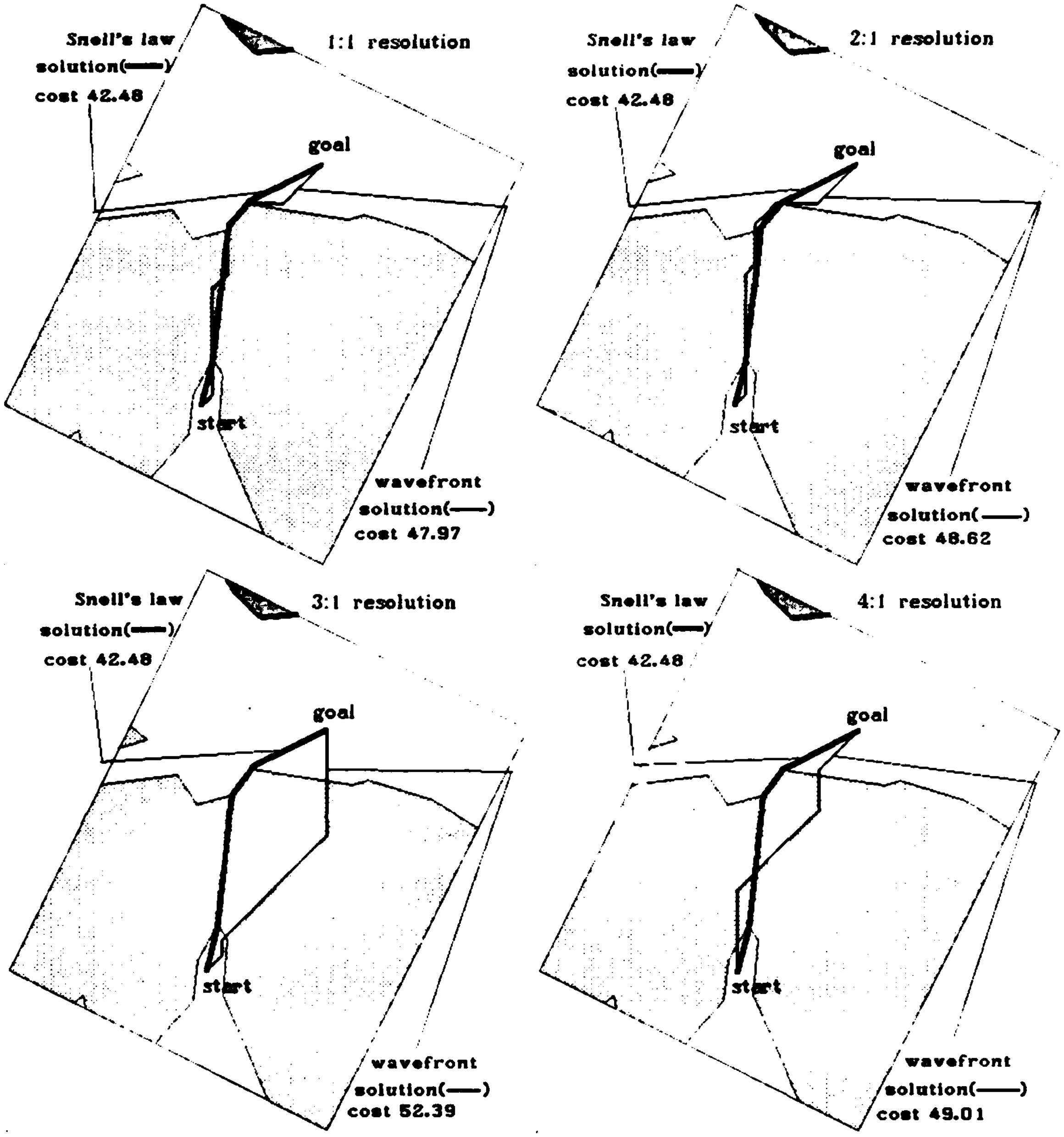


Figure 138. Problem 2, 1:1, 2:1, 3:1 and 4:1 Resolutions

found when using higher resolutions. The random nature of the accuracy is due to the (essentially) random manner in which the ternary-cost map is sampled. The cost rate for the point on the ternary-cost map having the same coordinates

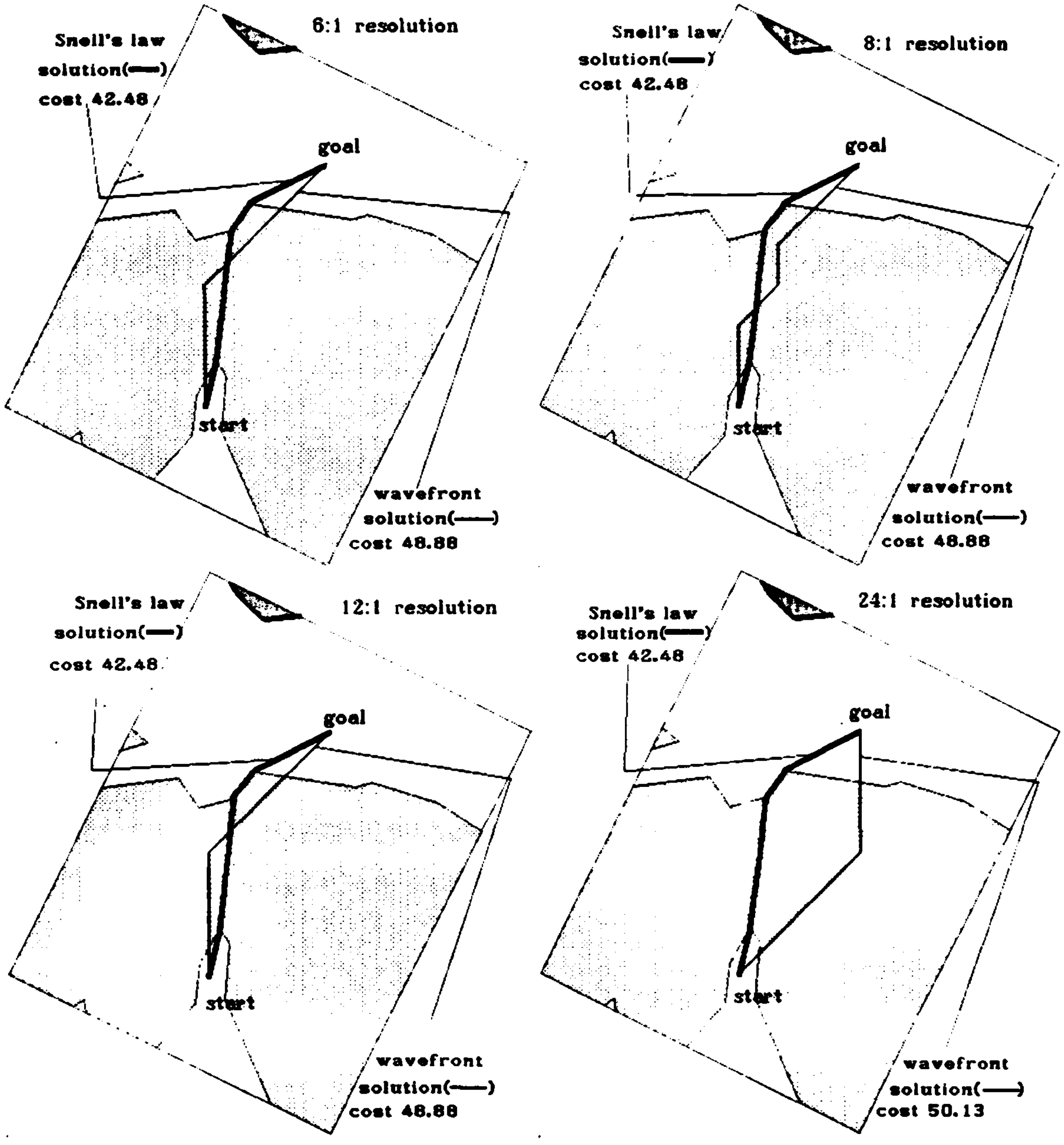


Figure 139. Problem 2, 6:1, 8:1, 12:1 and 24:1 Resolutions

as the lattice node is used as the cost rate for that node. In Figure 137 for example, the 24:1 resolution wavefront algorithm assumes that the solution path is comprised of entirely optimal-cost links since each lattice node on the solution path happened to have optimal cost. Thus, the effect of decreasing resolution to improve time performance has a random effect on the amount of error in solution paths. In fact, solution path feasibility may not be maintained at decreased resolutions. This can occur when two adjacent lattice nodes are separated on the area-cost map by a thin obstacle area (such as a fence). In this case, the lattice-based problem representation may not capture the fact that a path between these two adjacent lattice nodes is infeasible.

G. ALTERING COST-RATE RATIOS

We have also completed some testing where cost rate ratios are different than 2:1. The effect of this change on the wavefront algorithms is totally predictable. Lower cost ratios (i.e., 1.5:1) allow the algorithm to execute more quickly while higher ratios slow it down. This is most easily explained using the view of the wavefront algorithm as a simulation of the passage of time. Cells having higher costs require more time units to pass before the wavefront can be propagated through them. The converse is true for lower-cost cells.

The effect of altering cost ratios on the Snell's-law-based algorithm is not as obvious. In Table 20, we display the results of using the SL-Static algorithm to solve problems having some of the same start and goal locations on the same ternary-cost map while assigning different cost rates to the homogeneous-cost regions. The ratios 1.2:1, 2:1, and 6:1 were used. (Table 20 also includes data columns to show the number of wedges searched out of the number of wedges made (the "ws/wm" column). This data is analyzed below.) A primary effect of changing the cost ratio is that of altering thc size of the bounding box. In problems where either the start or goal (or both) is inside a high-cost region, at least some portion of the initial solution path must include a non-optimal-cost link. When the cost ratio is high (i.e., 6:1), this results in a larger bounding box, requiring more search effort because, in general, larger bounding boxes include more search points. A second effect of changing cost ratios has to do with the number of wedges that are searched. When the ratios are high, reflections occur more frequently. In general, this results in more empty wedges and thus fewer wedges are searched. These two effects of changing cost rate ratios interact with each other so that the performance of the Snell's-law-based algorithm is not predictable over similar problems involving homogeneous-cost regions with like geometry but different cost-rate ratios.

TABLE 20
ALTERING COST-RATE RATIOS

Prob	SL-Static 1.2:1		SL-Static 2:1		SL-Static 6:1	
#	ws/wm	Time (sec)	ws/wm	Time (sec)	ws/wm	Time (sec)
e	17/28	27.05	25/46	35.40	29/50	30.95
f	11/14	9.08	18/25	15.63	14/20	10.31
i	21/34	34.08	24/45	74.20	56/98	187.02
m	60/104	128.05	48/77	94.50	42/66	78.30
v	35/57	53.55	51/91	128.65	89/140	317.48
x	14/22	10.13	26/48	33.55	19/34	36.95
z	10/13	6.95	10/13	6.98	10/13	6.96
aa	17/26	11.66	15/26	17.06	18/32	19.50
ab	8/11	4.18	9/20	12.33	12/25	35.48
ag	44/83	131.82	43/82	124.53	29/77	103.88
ba	18/32	32.02	14/24	20.41	17/27	18.18

Table 21 allows a comparison of the SL-Dynamic and SL-Static algorithms on the same problems presented in Table 20. (Recall that SL-Dynamic uses stored information many times during algorithm execution and SL-Static uses such information only once to gain an initial solution.) Comparing the performance of these two algorithms on the same problems provides a measure of the benefits available by using stored path-cost information to calculate lower-bound cost evaluations. Note that dynamically using stored information does not always improve performance. In fact, only problems ag and ba reflect improved performance for the SL-Dynamic algorithm. However, the overhead involved in using stored information in this manner is small, a conclusion also supported by the data sample of Table 21. Thus, it is a good practice to use the information when it is available (from stored solutions to previously solved weighted-region problems for example). We also note that it is difficult to determine beforehand those cases where using such information results in performance improvements.

TABLE 21
SL-DYNAMIC AND SL-STATIC ALGORITHM PERFORMANCE

Problem	SL-Dynamic 2:1		SL-Static 2:1	
#	ws/wm	Time (sec)	ws/wm	Time (sec)
e	25/46	36.11	25/46	35.40
f	16/25	15.45	18/25	15.63
i	24/45	74.75	24/45	74.20
m	48/77	94.75	48/77	94.50
v	51/91	129.13	51/91	128.65
x	26/48	34.15	26/48	33.55
z	10/13	7.01	10/13	6.98
aa	15/26	17.20	15/26	17.06
ab	9/20	12.36	9/20	12.33
ag	40/77	117.51	43/82	124.53
ba	9/17	14.53	14/24	20.41

In a similar vein, we can compare the performance of the SL-None algorithm based on the data of Table 19. Recall that the SL-None algorithm does not use any stored cost information when determining an initial solution path. It solves the problem as if it were a binary problem and then computes the actual cost of the solution path on the ternary-cost map. As a result, SL-None initial solution paths tend to have a larger amount of high-cost path segments. Larger bounding boxes usually occur when using this scheme. In general, the bounding box areas depicted in Table 19 (as well as the number of search points) are high compared to the data presented in other tables. Clearly, the execution times reported in Table 19 are much higher as a result. Therefore, it is a good practice to use all available information when calculating an initial solution that will be used to construct the bounding box.

A final comparison we make is intended to provide some measure of the pruning criteria that have been implemented. We have counted the total number of wedges made and searched in each of the Snell's-law-based algorithm test problems. (The data point corresponding to problem ae was eliminated because only the two initial wedges were created and searched. The only search point in this problem was the goal itself; thus, there were no opportunities for wedge refinement or pruning.) The mean percentage of wedges searched out of those made is 59.7 per cent. There is a standard deviation of 8.82% in the data sample. The minimum percentage of wedges searched was 41.2% while a maximum of 81.8% of the wedges were examined. Thus, even the simple pruning criteria that have been implemented are able to eliminate from the search tree, on the average, almost half of the wedges created.

H. SUMMARY

Clearly, the Snell's-law-based algorithm requires less problem-description space than does the wavefront algorithm in almost every case. The Snell's-law-based method also solves weighted-region problems more quickly than does the

1:1 resolution wavefront. Lower-resolution wavefronts execute in less time, but sacrifice accuracy, and the amount of % added cost contained in their solution paths is essentially random. It is apparent that the time required by the Snell's-law-based algorithm depends primarily upon the number of search points that are inside the bounding box. Thus, the effort devoted to finding a good initial solution path is well spent since smaller bounding boxes usually contain fewer search points. Chapter VII provides a more detailed analysis of the time issue. A principal goal of the next chapter is the development of criteria that facilitate the selection of the most favorable algorithm, given a specific instance of the weighted-region problem.

VII. CONCLUSIONS

A. INTRODUCTION

In this chapter, we summarize the strengths and weaknesses of both the Snell's-law-based algorithm and the wavefront-propagation algorithm. The discussion is based upon properties of the two algorithms as demonstrated by their implementations used for testing in Chapter VI. We also address the development of criteria helpful in deciding which method to use when confronted with specific instances of the weighted-region problem. Finally, we briefly discuss related application areas for the Snell's-law-based algorithm and possible extensions to the technique.

B. WAVEFRONT PROPAGATION: STRENGTHS AND WEAKNESSES

A primary advantage of the wavefront-propagation algorithm lies in its simplicity. The method requires little more than applying a uniform-cost search to a lattice-like graph. In software engineering, the number of lines of code used to implement a strategy is sometimes used as a crude complexity measure for that strategy. We implemented the wavefront-propagation algorithm of Chapter VI in approximately 200 lines of Prolog. This compares to the more than 3000 lines of Prolog used to implement the Snell's-law-based strategy of Chapter VI.

The simplicity of the wavefront algorithm is chiefly responsible for establishing its stability; the algorithm always provides a solution path. Further, based on a simple metric, (the size of the bounding box determined from the cost of the initial solution) we can estimate, a priori, the time required by the algorithm to solve a given weighted-region problem. Since the algorithm only relies on simple arithmetic computations (i.e., no higher-order computations such as trigonometric calculations) numerical issues related to machine (or language) precision do not seriously affect wavefront propagation. (We note that simple arithmetic operations are also subject to precision and accuracy errors. However, their effect is small compared to the numerical issues that can affect the Snell's-law-based algorithm.) In short, the wavefront algorithm is simple and stable. It also has highly-predictable time-and-space requirements (as demonstrated below in Section VII.D).

The simplicity of the method has attendant drawbacks. The digital bias inherent in the lattice-based problem representation influences the technique so that wavefront propagation is, essentially, incapable of finding exact solutions to weighted-region problems. Also, the cost accuracy of wavefront-propagation solution paths is randomly influenced by representational resolution. One way of improving the accuracy of the wavefront method requires increasing the number of nodes (the resolution) in the lattice-based problem representation. Recall that increasing resolution by a factor of X increases the size of the lattice by X^2. This size increase translates directly into greater time-and-space requirements. Accuracy can also be improved by increasing the branching factor at each node in the lattice. While increased branching factors do not increase space requirements (since links are computed, not explicitly stored), the time required to search a graph with a higher branching factor does increase.

Even when using a very high resolution, digital bias can significantly affect solution path cost accuracy. Because of digital bias, the solution paths provided by wavefront propagation are optimal in the sense of a "Manhattan" or "city block" metric. Here, optimality is measured based on the search of a finite graph whose nodes represent every possible turn point that can be included on any path. The branching factor at each node determines a finite number of heading changes that any path can take at each turn point. We have seen that there is, essentially, an infinite number of possible turn points and heading changes that must be considered when attempting to find optimal-cost solution paths for the weighted-region problem. Thus, no finite amount of increased resolution or branching factor can overcome the inaccuracy inherent in a search strategy that is based on a "Manhattan" metric.

In Section II.E.2.c we described an implementation of the wavefront algorithm that exploits parallelism. However, the advantages of parallel machines are not realized until a relatively large number of processors are in use, although large numbers of processors can greatly improve the time performance of the algorithm. Mesh-connected architectures offer the potential for the development of wavefront-propagation algorithms having linear time complexity (with respect to the number of lattice nodes on a solution path).

A final difficulty with the wavefront algorithm concerns the costs assigned to the lattice nodes describing the area-cost map. We have noted that this is a difficult problem because of the resolution and cost-aggregation issues that must be resolved. It seems that the problem of assigning aggregate cost rates to lattice nodes admits more approximations into the wavefront algorithm problem representation. More approximations can mean that solution paths will be less accurate. Some simple cost-aggregation strategies (such as point sampling) may not even ensure path feasibility.

In summary, the wavefront algorithm is simple and robust. However, it can have large time-and-space requirements (when compared to the Snell's-law-based algorithm for example). If decreased resolution is used to lessen these requirements, unpredictable cost errors in solution paths can occur. When parallelism is exploited to reduce execution time requirements, the digital bias problem still remains to adversely affect solution accuracy. Two distinct sources of error in the wavefront-propagation algorithm are based on the problem representation. The number of nodes in the lattice as well as the branching factor at each node affect the accuracy of wavefront-propagation solution paths.

C. SNELL'S-LAW-BASED ALGORITHM: STRENGTHS AND WEAKNESSES

A principal advantage of the Snell's-law-based method is that it provides more accurate (i.e., lower cost) solution paths than those found by the wavefront-propagation algorithm. (Recall from Chapter VI that, in every test problem, the Snell's-law-based method found a lower-cost solution path than did the wavefront algorithm.) Moreover, in comparison to the wavefront method, path-cost accuracy is not achieved with large execution-time requirements. The testing reported in Chapter VI also showed that the Snell's-law-based method generally had lower problem-description space requirements than the wavefront-propagation algorithm. The average-case time performance of the Snell's-law-based algorithm seems to be quadratic in the number of search points inside the bounding box (as illustrated in Section VII.D, Figure 140B).

The Snell's-law-based strategy is suitable for parallel implementation. Recall the tree of wedges illustrated in Figure 124B. The iterative solution of Snell's law within each wedge is independent. Thus, a different processor can be assigned to iteratively solve Snell's-law problems within each newly-created wedge. The only inter-processor communication necessary during search involves the upper bound on the cost of the optimal-cost solution path. If sufficient processors are available, pruning based on an upper bound can be abandoned and an exhaustive search conducted so that every known wedge is searched. If an upper bound is not used, then it never needs to be updated so it never needs to be written. Thus, the locking protocols [Ref. 54] needed to allow multiple processors to write data are superfluous and no communication is necessary between processors. (Note that no inter-processor communication is required for each processor to read the area-cost map since read operations can be conducted in parallel without locking.) Also, even low levels of parallelism can be exploited by the Snell's-law-based algorithm. For example, if two processors are available, one of the two initial wedges can be assigned to each of them.

The Snell's-law-based algorithm does not suffer any problems related to digital bias. It is always able to find straight-line solution paths and describe them by only two path endpoints. Thus, the Snell's-law-based method also provides minimal descriptions (in terms of space) of solution paths. Wavefront solution paths are described as a series of points, one for each lattice node on the path (according to the resolution in use). Snell's-law-based solution paths only include the coordinates of the start, goal, and any intermediate turn points on the optimal-cost path between them.

Resolution of the area-cost map can be used to define the desired cost accuracy of solution paths. As an example, modeling a jagged-boundary region by a polygon with only a few vertices can result in an approximate solution, relatively quickly achieved. A more accurate model of the region (i.e., a polygon with more vertices) supports a more accurate solution, arrived at more slowly. Thus, we can predict the effects of altering the resolution of the homogeneous-cost region problem representation. A higher resolution representation generally results in a more accurate solution. Also, as should be expected, solution paths based on high-resolution representations require more time and space to be achieved.

The more intelligent problem representation used by the Snell's-law-based method results in several benefits. It not only requires less problem-description space (on the average, as shown in Chapter VI) and eliminates digital bias, but it eliminates the cost-aggregation problem as well. When using a homogeneous-cost regions to represent the problem, it is simple to ensure that important areas of the area-cost map are not overlooked. Moreover, it is also simple to use different resolutions for different parts of the same area-cost map. That is, if some portions of the area-cost map seem most important, they can be described by polygons having many vertices. Other, less important areas can be grossly modeled by polygons with fewer vertices.

The Snell's-law-based algorithm can also adapt to dynamically-changing map information. For example, suppose that an instance of a weighted-region problem has been solved. As a side effect of solving the problem, the area-cost map is divided into a set of wedges. It could occur that during execution of a planned route, an agent might update the information on the area-cost map, based on locally-sensed information. Suppose that an area originally thought to be traversable at high cost is found to be an optimal-cost region. In this case, only those wedges that intersected the original (erroneously classified) high-cost region need be considered to find a new optimal-cost path based on updated information. Those wedges that did not intersect the newly-classified area on the original area-cost map are clearly not affected by that area's elimination from the map. Assuming that a solution path has already been found, the wavefront algorithm can only utilize updated map information by re-solving the entire problem again. This is because the wavefront method searches over the area-cost map as an entity while the Snell's-law-based strategy divides the map into independent wedges. A similar characterization of the two algorithms arises when a previously unknown high-cost region is found to be on the optimal-cost solution path. Thus, because the Snell's-law-based approach supports the division of the map into independent areas (the wedges), it is able to reason about changing map information more efficiently.

Recall that the ADS wavefront implementation (Section III.E.2.c, [Ref. 36]) actually finds the optimal-cost path (in terms of the Manhattan metric) from every point on the map to the goal. This can be useful when an agent wanders off-course during the execution of a planned route. In this event, the agent need only locate itself in the correct map cell and "look up" the previously computed optimal-cost path from that cell to the goal. A similar "look up" operation is not possible when the goal location changes (perhaps a new goal is assigned while the agent is enroute). Again, the Snell's-law-based method can exploit its division of the map into wedges to efficiently handle these situations. If the goal's location or the agent's location is changed but is still within the wedge containing the solution path, the solution path can be locally adjusted (within the wedge). If the location change moves either point to a new wedge, only those wedges that contain that point need be examined to find a new optimal-cost path.

The accuracy, relative speed, and robustness of the Snell's-law-based algorithm come at the expense of its complexity. The algorithm is more difficult to comprehend and implement than the wavefront approach. In many ways, the algorithm is made even more complex by numerical issues. There must be provisions for dealing with boundary cases of trigonometric routines and line-intersection routines. Because of this, numerical issues can slow the algorithm's convergence to a solution. A final problem with the Snell's-law-based approach is that its time and space requirements are not highly predictable (in comparison to those of the wavefront algorithm). Opportunities for pruning occur, more or less, randomly. Thus, it is difficult to determine, a priori, highly-accurate estimates of the time required by the Snell's-law-based algorithm to solve a specific weighted-region problem. Also, we have not been able to establish polynomial complexity bounds (in time or space) for the Snell's-law-based algorithm in worst-case situations. However, this worst-case bound may not be very meaningful. Again we draw an analogy between our Snell's-law-based algorithm and the simplex algorithm used to solve linear programming problems. Both algorithms have exponential worst-case time bounds. However, both algorithms perform well in the average case.

D. SELECTING THE APPROPRIATE SOLUTION METHOD

Both the wavefront-propagation technique and the Snell's-law-based algorithm have advantages and disadvantages. Often, one method is better suited to a specific requirement than the other. As an example, when accuracy of the solution is of paramount importance, the Snell's-law-based approach is most appropriate. Under any circumstances, the digital bias inherent in the problem representation used by the wavefront-propagation algorithms prevents them from achieving highly cost-accurate solution paths for weighted-region problems.

The Snell's-law-based approach is the most appropriate method in several circumstances. When many problems involving the same map must be solved, the Snell's-law-based method is the technique of choice since it can utilize a primitive form of learning to improve its performance over time. While the wavefront-propagation algorithm can use known solutions to aid in finding initial solutions, the Snell's-law-based algorithm can use known solutions (even after the initial solution path has been found) to compute lower-bound cost evaluations, enhancing pruning capabilities. The wavefront algorithm does no pruning after initialization so it does not fully exploit stored information. If a low level of parallelism is available, the Snell's-law-based method is preferable. The wavefront technique can utilize highly-parallel architectures, but its performance is not much enhanced by the availability of only a few additional processors. When space constraints are severe, the Snell's-law-based method is preferable since it generally requires less problem-description space. The Snell's-law-based method is also the technique of choice when the map information is likely to be changing, dynamically. Time constraints can also determine the most appropriate method.

The wavefront algorithm can require less time when the area of the bounding box is small or when there is a large number of region vertices to be considered. Figures 140A and 140B make this trend more apparent. In Figure 140A we have plotted the time required by the wavefront-propagation algorithm to reach a solution (on the vertical axis) versus the area of the bounding box (on the horizontal axis) for each problem in the set of test problems reported in Chapter VI. Recall that the area of the bounding box is approximately equivalent to the number of nodes in the lattice-based problem representation used by the wavefront-propagation algorithm. The unshaded circles represent 1:1 resolution wavefront performance while the darkened circles depict 2:1 resolution wavefront performance. (Also, recall that solving a problem at 1:1 resolution requires 4 times as many lattice nodes as solving the same problem at a 2:1 resolution. The data points in Figure 140A reflect this fact as the darkened circles are all lower on the area scale than the open circles, even though they represent performance on some of the same start-to-goal problems.) We have fit a straight line to the data using least-squares regression. Figure 140B is a similar illustration derived from the Snell's-law-based algorithm performance on the same set of test problems. In Figure 140B, the time required to solve the problems is plotted along the horizontal axis while the square of the number of region vertices located inside the bounding box is plotted on the vertical axis. Note that this data supports a quadratic average-case time complexity for the Snell's-law-based algorithm (with respect to the number of vertices in the bounding box).

The regression lines in Figure 140A and 140B can be used to predict the (approximate) solution-time requirement of either algorithm once the bounding

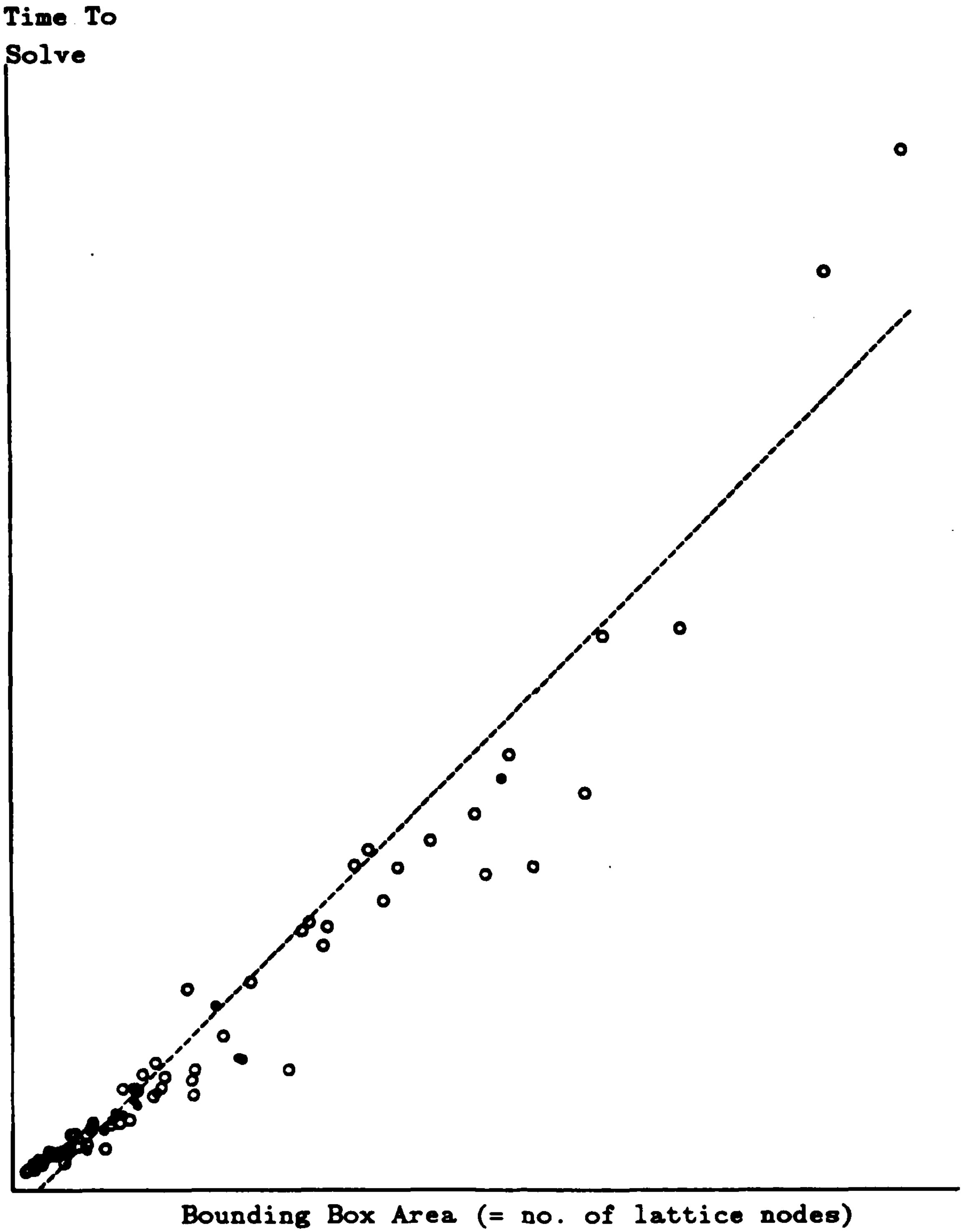


Figure 140A. Wavefront-Propagation Performance

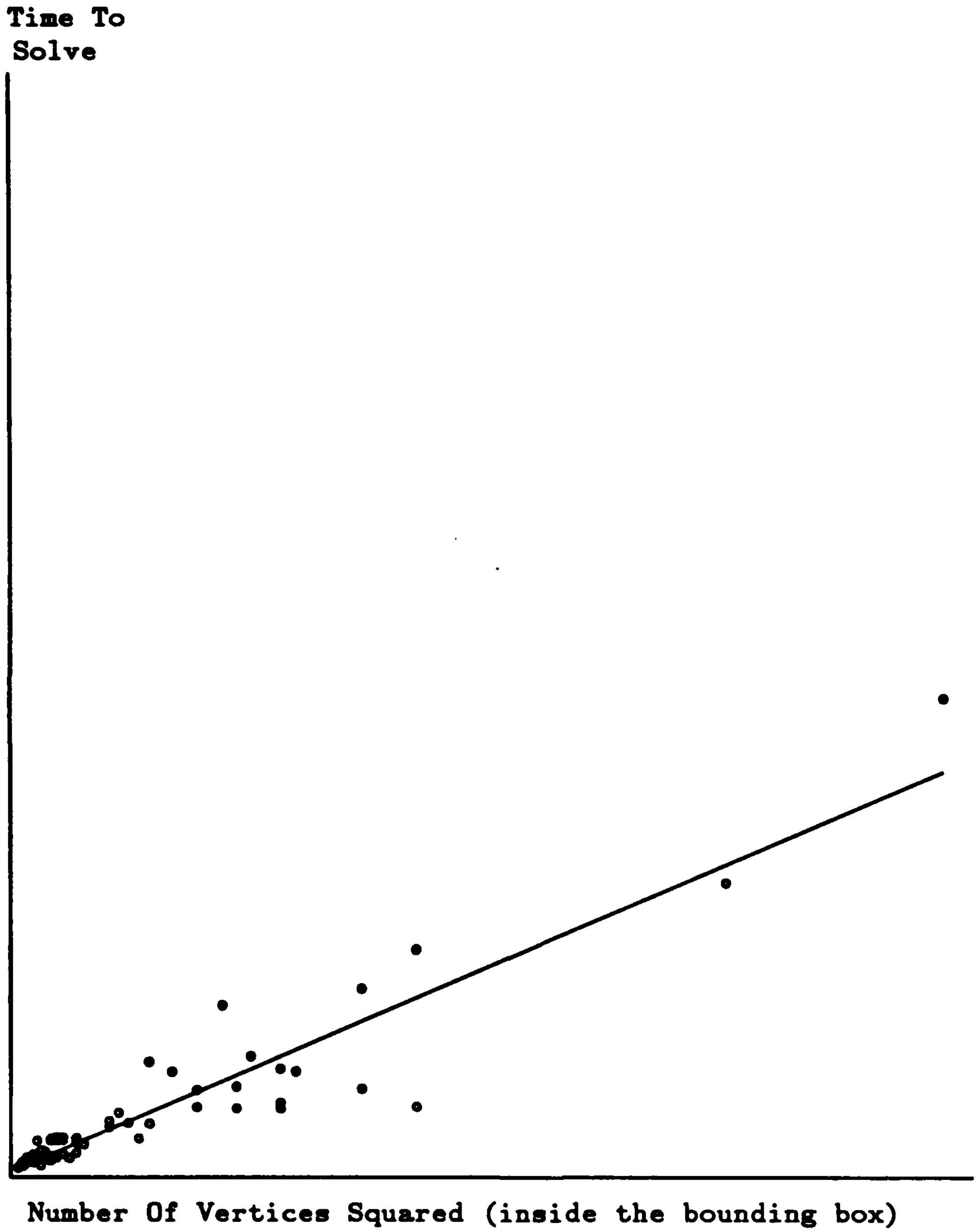


Figure 140B. Snell's-Law-Based Algorithm Performance

box has been constructed. Specifically, the area of the bounding box (measured in the same units as the resolution of the lattice-based problem representation) predicts wavefront-propagation time while the square of the number of region vertices inside the bounding box predicts time requirements for the Snell's-law-based algorithm. Thus, when the time required to solve a specific instance of the weighted-region problem is of paramount importance, we can use these regression lines to select the technique that promises a solution in the least amount of time. The resolution (or grid size) of the lattice used by wavefront propagation is immaterial in this comparison since the bounding box area must be expressed in the same units as the lattice resolution. The direct comparison value of the linear-regression predictors is more clearly seen in Figure 141 where the two regression lines appear together. The direct comparison is valid since Figures 140A and 140B are based on exactly the same set of test problems.

Issues of accuracy aside, the wavefront algorithm may turn out to be the technique of choice based on time-requirement predictions, particularly if a low-resolution wavefront can be applied. The technique can also be most appropriate when stability and simplicity are desired. The wavefront can also be the best strategy when region vertices are tightly grouped in small areas, a situation that frequently gives rise to numerical problems for the Snell's-law-based algorithm.

In summary, the Snell's-law-based method provides less costly (and thus more accurate) solution paths than does the wavefront-propagation algorithm. When region vertex density (within the bounding box) is relatively low, it is also the most time and space efficient method, especially when the bounding box includes a large area. However, the Snell's-law-based method has a higher worst-case

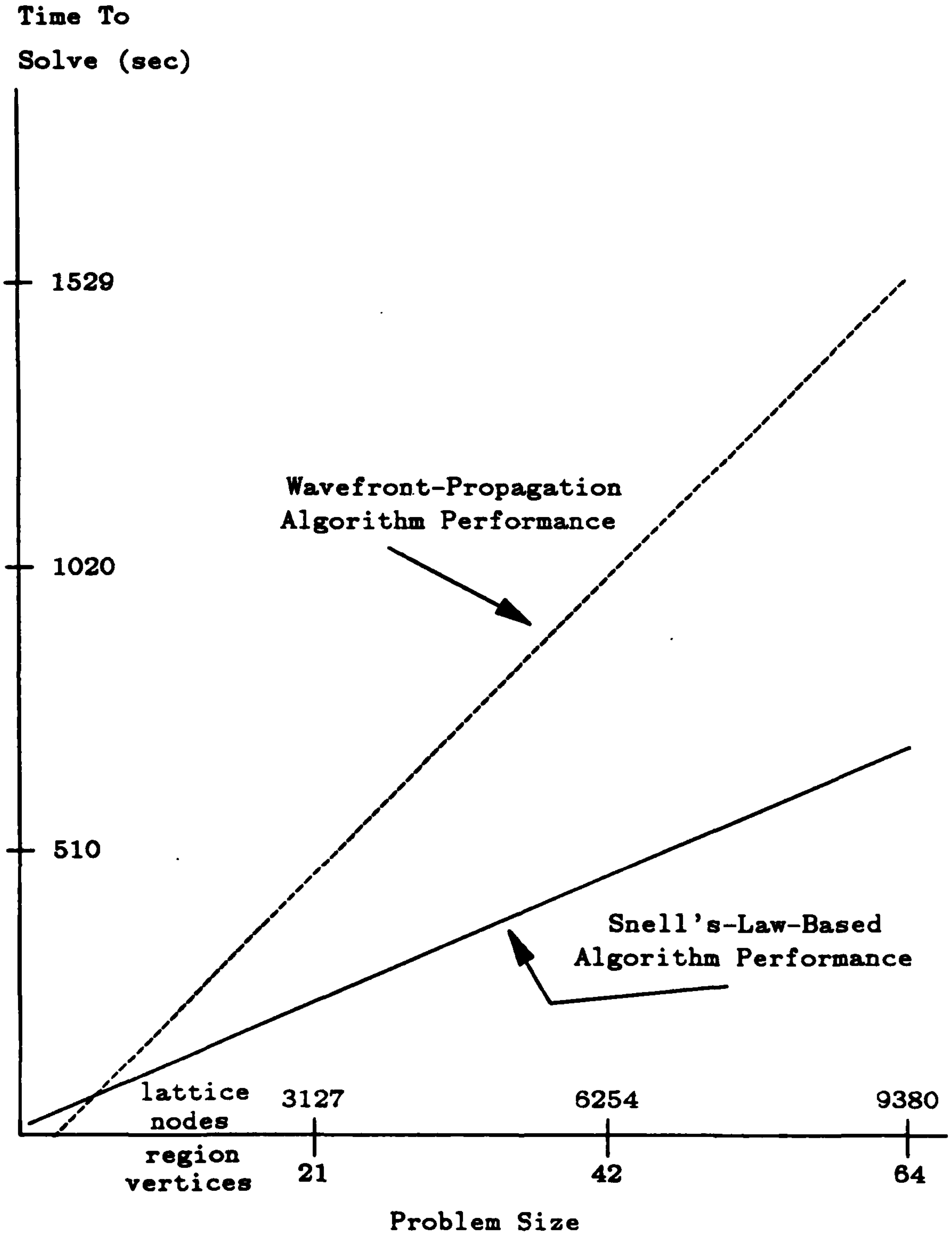


Figure 141. Regression Line Comparison

complexity bound than the wavefront algorithm. Thus, for problems where an approximate solution is sufficient, wavefront propagation can be the technique of choice.

E. POSSIBLE EXTENSIONS

Our implementation of the Snell's-law-based algorithm is a prototype. It remains to move beyond the ternary-cost map restriction. We would need to recode the algorithm to correctly solve weighted-region problems on area-cost maps featuring different cost regions which abut each other (i.e., each high-cost region should not be required to be surrounded by an optimal-cost region as described in Section VI.B.2). These alterations can be accommodated into the program logic with little effort.

There are many opportunities to enhance the pruning abilities of the Snell's-law-based algorithm. Currently, only a very primitive form of learning is possible. The algorithm could make better use of stored information, much as humans are able to use previously known routes to aid in selecting new ones. As an example, we could enhance the algorithm with the ability to learn that some areas of the map (almost) never contain portions of optimal-cost solution paths and eliminate (or postpone the search of) these areas. Clearly, a more powerful learning component can lower the time requirements of the Snell's-law-based algorithm. There are some issues involved in indexing the most helpful information that must be overcome (this is similar to the indexing issue in the MOLGEN program discussed in Section II.C.5). Also, it should be possible to use previously-computed wedges as well as path costs to decrease time requirements. Sometimes, the same wedge is reconstructed during the solution of different start-to-goal problems that come from the same area-cost map. Saving previously computed wedges could reduce computational effort in such instances.

In many cases, the initial solution is actually the optimal-cost solution or is physically very close to it. This occurs most often when cost-rate ratios are high (6:1 for example). There should be some way to detect the optimality of initial solutions, without resort to search (as is currently the case). (As an example, in the Appendix, the first demonstration problem relies on the optimal-cost solution path as the initial solution. In this case, the algorithm uses search only to verify the optimality of the initial solution.) Developing such criteria can improve the performance of both the wavefront-propagation and the Snell's-law-based algorithms. In the same vein, the development of simple methods to achieve lower-cost initial solutions is a worthwhile extension. Lower-cost initial solutions result in smaller bounding boxes that can include fewer region vertices.

Finally, the development of a system that dynamically selects the best algorithm for application in a specific situation is desirable. It is possible to intermix use of the Snell's-law-based algorithm and the wavefront-propagation algorithm during the solution of a single problem. A wavefront can be propagated only within a wedge as an example. Selection of a method to apply can be based on knowledge about wedges: how large they are and how many search points they contain. There may also be a method to use a fast, low-resolution wavefront to limit the search space for an accurate Snell's-law-based algorithm.

F. OTHER APPLICATION AREAS

The Snell's-law-based algorithm we have described is intended for use in planning optimal-cost land routes. We note that cost is generic. The Snell's-law-based algorithm assumes an appropriate area-cost map, without regard to the cost measure or the agent for which the route is to be planned. Thus the algorithm can also be applied, without change, to plan the location of overland pipelines or road networks, provided with appropriately classified area-cost maps. In general, the algorithm is applicable to any problem where the solution is the location of an optimal-cost route through a two-dimensional space of homogeneous-cost regions.

The algorithm can be slightly modified so that it returns a set of feasible routes through the space represented by the area-cost map. In general, the algorithm computes multiple solution paths, the least-cost of which is returned as the optimal-cost solution path. When multiple paths are required, the algorithm can be used by "turning off" pruning, causing all feasible solution paths to be found. Similarly, the *n* best paths can be found (when at least *n* feasible paths exist). In this form, the algorithm is suitable for finding multiple avenues of approach to a single goal location.

The algorithm can also be used to find the shortest distance between regions (i.e., polygons). The CDA, reported in [Ref. 42] has been used for this purpose. To achieve these results, the start and goal are embedded in "zero-cost" regions such that the cost accrued by traveling from anywhere in the interior of the region to any of the region boundaries is zero. Given this configuration, the start-to-goal solution path includes the shortest (weighted) distance path between the two zero-cost region polygons [Ref. 3].

Ray tracing is a basic operation for the Snell's-law-based algorithm. Thus some portions of the algorithm are suitable for use in applications concerned with tracing the paths of light rays through translucent materials. Normally, lighting and shading algorithms compute the light intensity (from a point source) for each screen pixel on a computer graphics monitor. We can apply "wedges of light" to this task, resulting in groupings of pixels that have a uniform intensity due to a light source located at the wedge tip. Some modifications are necessary for the algorithm to fit the application. However, there are key similarities in the two problem areas.

G. SUMMARY

Snell's law can be applied to the weighted-region problem. This principle of optics serves as a local optimality criterion, much as the straight-line hypothesis has been employed by the VGraph algorithm in solving binary-case problems. Use of Snell's law also facilitates a more intelligent problem representation that describes regions, not arbitrary grid cells in a lattice. Applying Snell's law to this type of problem representation results in an algorithm that does not suffer many of the deficiencies inherent in the wavefront-propagation technique. The Snell's-law-based algorithm provides more accurate solution paths (at a lower time cost) than the wavefront-propagation algorithm. In general, the Snell's-law-based algorithm also requires less problem-description space. When the time required to solve a specific instance of the weighted-region problem is of paramount importance, we can select the technique that promises a solution in the least amount of time.

The Snell's-law-based algorithm we have presented employs several ideas commonly used in artificial intelligence. An informed strategy (A*) is used to conduct a search over a dynamically created graph that is based on a specific weighted-region problem. This graph is created based upon an appropriate problem representation that models regions, not discrete points. Previously computed solutions can be used to limit the search effort, both globally and locally. Recursive problem decomposition is applied (at diffraction vertices). Heuristics are used to order search efforts (through the agendas).

Thus the algorithm relies on interdisciplinary precepts. A principle from optics serves as the local optimality criterion; optimization is used to constrain search; computer-science techniques are used. These principles are combined to form an algorithm that has a firm mathematical basis and is capable of providing accurate solutions to instances of the weighted-region problem while often conserving both time and space.

LIST OF REFERENCES

1. Brady, M., "Artificial Intelligence and Robotics", *Artificial Intelligence,* Elsevier Science Publishing, Vol. 26 (1985), pp. 79 - 121.

2. Charniak, E. and McDermott, D., *Introduction to Artificial Intelligence,* Addison-Wesley, Reading, 1985.

3. Mitchell, J.S.B. and Papadimitriou, C.H., *The Weighted Region Problem,* Tech. Report, Dept. of Operations Research, Stanford University, October 1985 (revised July 1986).

4. Pars, L.A., *An Introduction to the Calculus of Variations,* Heinemann Educational Books, Ltd., London, 1962.

5. Winston, P., *Artificial Intelligence,* Addison-Wesley, Reading, 1984.

6. Pearl, J., *Heuristics, Intelligent Search Strategies for Computer Problem Solving,* Addison-Wesley, Reading, 1984.

7. Simon, H.A., *The Sciences of the Artificial,* Second Edition, The MIT Press, Cambridge, 1981.

8. Rich, E., *Artificial Intelligence,* McGraw-Hill, New York, 1983.

9. Dijkstra, E., "A Note on Two Problems In Connection With Graphs", *Numerische Mathematik,* 1(1959), pp. 269 - 271.

10. Hart, P.E., Nilsson, N.J., and Rạphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", *IEEE Transactions on Systems Science and Cybernetics,* SSC-4(2), 1968, pp. 100 - 107.

11. Barr, A. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence, Volume I,* William Kaufmann, Inc., Los Altos, 1981.

12. Cohen, P.R. and Feigenbaum, E.A., *The Handbook of Artificial Intelligence, Volume III,* William Kaufmann, Inc., Los Altos, 1982.

13. Bazara, M. and Jarvis, J., *Linear Programming and Network Flows,* John Wiley and Sons, New York, 1977.

14. Sedgewick, R., *Algorithms,* Addison-Wesley, Reading, 1983.

15. Feigenbaum, E.A. and Feldman, J. (Eds), *Computers and Thought,* McGraw-Hill. New York, 1963.

16. Hayes-Roth. B. and Hayes-Roth, F., "A Cognitive Model of Planning", *Cognitive Science.* Vol. 3(1979), pp. 275 - 310.

17. Friedland, P.E., *Knowledge-Based Experiment Design in Molecular Genetics,* (Doctoral dissertation), Rep. No. 79-771, Computer Science Dept., Stanford University, 1979.

18. Sacerdoti, E.P., *A Structure for Plans and Behavior,* (Doctoral dissertation), Tech. Note 109. AI Center, SRI International, Inc., Menlo Park, 1975.

19. Lozano-Perez, T. and Wesley, M., "An Algorithm for Planning Collision Free Paths Among Polyhedral Obstacles", *CACM,* Vol. 22, No. 10, October 1979, pp. 165 - 175.

20. Nevatia, R., *Machine Perception,* Prentice-Hall, Englewood Cliffs, 1982.

21. Brooks, R.A., "Solving the Find-Path Problem by Good Representation of Free Space", *IEEE Transactions on Systems, Man and Cybernetics,* SMC-13(3), 1983, pp. 190 - 197.

22. Thorpe, C., "Path Relaxation: Path Planning for a Mobile Robot", *Proceedings of the National Conference on Artificial Intelligence,* Austin, Texas, 1984.

23. Buckley, C. and Leifer, L., "A Proximity Metric for Continuum Path Planning", *Proceedings of IJCAI-9,* Los Angeles, August 1985.

24. Gillmore, J. and Semenco, A., "Terrain Navigation Through Knowledge Based Route Planning", *Proceedings of IJCAI-9,* Los Angeles, August 1985.

25. Giralt, G., Sobek, R., and Chatila, R., "A Multi-Level Planning and Navigation System for a Mobile Robot", *Proceedings of IJCAI-6,* August 1979.

26. Moravec, H., "The CMU Rover", *Proceedings of AAAI-82,* August 1982.

27. Richbourg, R. et al, "Exploiting Capability Constraints to Solve Global, Two Dimensional Path Planning Problems", *Proceedings, IEEE International Conference on Robotics and Automation,* San Francisco, April 1986, pp. 91 - 95.

28. Thompson, A., "The Navigation System of the JPL Robot", *Proceedings, IJCAI-5,* 1977.

29. Witkowski, C., "A Parallel Processor Algorithm for Robot Route Planning", *Proceedings, IJCAI-8,* Karlsruhe, FRG, August 1983.

30. Quek, F.K.H., Franklin, R.F. and Pont, F., "A Decision System for Autonomous Robot Navigation Over Rough Terrain", *SPIE, Intelligent Robots and Computer Vision,* Vol. 579, 1985, pp. 377 - 389.

31. Rowe, N.C., "Roads, Rivers, and Rocks: Optimal Two-Dimensional Route Planning Around Linear Features, Technical Report, Naval Postgraduate School, Monterey, Ca., May 1987.

32. Keirsey, D. and Mitchell, J.S.B., "Planning Strategic Paths Through Variable Terrain Data", *SPIE, Vol. 485, Applications of Artificial Intelligence,* 1984.

33. Keirsey, D., Mitchell, J.S.B., Payton, D. and Preyss, E., "Path Planning for Automomous Vehicles", *SPIE, Vol. 485, Applications of Artificial Intelligence,* 1984.

34. Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P., "Optimization by Simulated Annealing", *Science,* Vol. 220, No. 4598, 13 May 1983, pp. 671 - 680.

35. Lindsay, C., "Automatic Planning of Safe and Efficient Robot Paths Using an Octree Representation of a Configured Space", presented at the IEEE International Conference on Robotics and Automation, 2 April 1987, Raleigh, N.C.

36. Linden, T.A., Marsh, J.P. and Dove, D.L., "Architecture and Early Experience With Planning for the ALV", *Proceedings, IEEE International Conference on Robotics and Automation,* San Francisco, April 1985, pp. 2035 - 2042.

37. Hillis, W.D., "The Connection Machine: A Computer Architecture Based On Cellular Automata", pp. 491 - 506, *Computers for Artificial Intelligence Applications,* Wah, B. and Li, G-J, eds., IEEE Computer Society Press, Washington, 1986.

38. Jorgenson, C., "Robot Navigation Using Neural Networks", presented at the Workshop on Advanced Computer Architectures for Robotics and Machine Intelligence: Neural Networks and Neurocomputers, IEEE International Conference on Robotics and Automation, 3 April 1987, Raleigh, N.C.

39. Faulkner, F., "Optimal Ship Routing", *Navigation,* Vol. 10, No. 1, Winter 1963, pp. 351 - 367.

40. Finney, R. and Thomas, G., *Calculus and Analytic Geometry, 5th Edition,* Addison-Wesley, Reading, 1981.

41. Chavez, R. and Meystel, A., "A Structure of Intelligence for an Autonomous Vehicle", *Proceedings, IEEE International Conference on Robotics,* Atlanta, 1984, pp. 584 - 591.

42. Mitchell, J.S.B., "Planning Shortest Paths", (Doctoral dissertation), Dept. of Operations Research, Stanford University, 1986.

43. Stavroudis, O.N., *The Optics of Rays, Wavefronts and Caustics,* Academic Press, New York, 1972.

44. McGhee, R. and Iswandi, G., "Adaptive Locomotion of a Multilegged Robot Over Rough Terrain", *IEEE Transactions on Systems, Man and Cybernetics,* April 1979.

45. Michalski, R.S., Carbonell, J.G. and Mitchell, T.M., eds., *Machine Learning, An Artificial Intelligence Approach,* Tioga, Palo Alto, 1983.

46. Knuth, D.E., *The Art of Computer Programming, Volume III, Sorting and Searching,* Addison-Wesley, Reading, 1973.

47. Pavlidis, T., *Algorithms for Graphics and Image Processing,* Computer Science Press, Rockville, 1982.

48. Bowyer, A. and Woodwark, J., *A Programmer's Geometry,* Butterworths, London, 1983.

49. Press, W.H. et al, *Numerical Recipes, The Art of Scientific Computing,* Cambridge University Press, Cambridge, 1986, pp. 241 - 248.

50. Klee, V.L. and Minty, G.J., "How Good Is The Simplex Algorithm?", Mathematical Note No. 63, Mathematics Research Laboratory, Boeing Scientific Research Labs, Feb, 1970, reprinted in Shinsa, O. (ed.), *Inequalities III,* Academic Press, New York, 1972, pp. 159 - 179.

51. Khachiyan, L.G., "A Polynomial Algorithm In Linear Programming", Doklady Akademiia Nauk SSSR 244is (1979), pp. 1093 - 1096, translated in *Soviet Mathematics Doklady,* Vol. 20, No. 1, 1979, pp. 191 - 194.

52. Clocksin, W.F. and Mellish, C.S., *Programming in Prolog,* Springer-Verlag, New York, 1981.

53. Shapiro, E. and Sterling, L., *The Art of Prolog,* The MIT Press, Cambridge, 1986.

54. Date, C.J., *An Introduction to Database Systems, Volume 1,* (4th edition), Addison-Wesley, Reading, 1986.

APPENDIX (DEMONSTRATION)

Section V.J included a demonstration of our Snell's-law-based algorithm as it found the optimal-cost path between a start and goal both embedded in high-cost regions. We now illustrate the solution process of the algorithm as it solves two new problems. In the first demonstration problem, both the start and goal are located in low-cost regions. In the second problem, one point (the start) is inside a high-cost region while the other is in a low-cost area. Both problems are taken from Map1 and feature a 2:1 ratio between the cost rates of the traversable regions. The SL-Static algorithm was used to solve both problems. (Note that all figures and tables are located at the rear of the appendix, after page 378.)

In the first problem, the start is located at coordinates (56,38) and the goal at coordinates (31,73). The problem is illustrated in Figure 142. This figure also depicts the initial solution path and the bounding box that was created based on the cost of this path. For the first problem, the initial solution turns out to be the optimal-cost solution path. Table 22 provides a wedge identification ("Wedge ID") that can be used to correlate the wedge tree of Figures 143A and 143B to the wedge illustrations in Figures 144 through 158. The first column of Table 22 lists the Wedge ID, exactly as used in the wedge tree of Figure 143A. These identifications have the form "WX" or "RWX" where "X" is an integer. A wedge denoted as "RWX" is a reflection wedge while those denoted "WX" are regular (non-reflection) wedges. Note that the reflection wedges are listed at the end of Table 22 (which spans more than one page).

The second column of Table 22 lists the left and right wedge-defining Snell's-law paths as a series of turn points. The path that defines the wedge left boundary is listed above the path defining the right wedge boundary. Each turn point has a letter designation. Path intersections with the bounding box are denoted as "zX" where "X" is an integer. This same convention for path description is used in Figures 144 through 158. Table 22 also includes the A* evaluation of each wedge in the form "g(W)+h(W)=f(W)". (We use this form for brevity in table headings.) Recall that g(W) is a known cost associated with the wedge approach path, h(W) is a lower-bound cost evaluation of a start-to-goal path within the wedge, and f(W) is the total-cost evaluation for the wedge (which is also a lower bound).

Figures 143A and 143B depict the wedge search tree. Each node lists the Wedge ID (from Table 22) and the wedge's total-cost evaluation (i.e., the f(W) value). When a circled number appears above the Wedge ID, it indicates that the wedge was searched and in what order it was removed from an agenda. Note that some wedges in the tree have a branching factor of 4 due to reflection wedges. These wedges are indicated by appropriate Wedge ID's (starting with an "R") and by a dashed line showing ancestry.

Some nodes in the tree do not have all 4 possible sub-wedges (or child wedges). For example, the left child of wedge W7 (Figure 143A) was not created because it would have overlapping left and right wedge-defining Snell's-law paths. This can be seen in Figure 150. The left sub-wedge would have path S-i-l as both the left and right wedge-defining path, creating an empty wedge. Thus, the algorithm did not expend the effort to create this sub-wedge.

Similarly, some wedges do not have specific child sub-wedges due to obstacles. For example, in Figure 144, a left child of wedge W1 was not created because of the obstacle boundary between points a and b. Since this wedge has no search points between the wedge tip (S) and the obstacle boundary, it can not be further refined. Also, any path through this wedge would have to end at the obstacle boundary. Thus, this wedge cannot contain an optimal-cost solution path and does not need to be created.

Finally, the wedge tree includes some nodes that are described as "No Middle Sub-wedge". Most often, overlapping left and right wedge boundaries create these empty middle sub-wedges, although obstacles can also affect the situation. Other than for these reasons, all nodes have three (or four in reflection cases) child sub-wedges. Figures 143A and 143B use the convention of listing the left, middle and right sub-wedges in that order from left to right.

Figures 144 through 158 show the parent wedge (in the upper left corner of each figure) and the solution path to the closest unsolved search point within the parent wedge (in the upper middle inset of each figure). When a reflection sub-wedge can be created, it is shown as the upper right inset of the figure. The lower half of each figure shows the three child sub-wedges that are created based on the solution path. Figures 144 through 158 depict exactly the same homogeneous-cost region geometry as that enclosed by the bounding box of Figure 142. However, the regions have been scaled and rotated so that all six insets for each figure can be placed on a single page. The darkly-shaded polygons depict obstacle areas while the lightly-shaded areas are high-cost, traversable regions. The unshaded background is the low-cost, traversable area. Note that only one start-to-goal solution path is found for the first problem (as illustrated in Figure 158) and it is the same path that was used as the initial solution. Also, only one reflection wedge is formed during the solution process (wedge RW1, Figure 156) and it is pruned immediately based on the upper bound for the cost of the optimal solution path.

The second problem features a start point located inside a high-cost region but, otherwise, has very similar geometry to the first problem. However, the initial solution does not have optimal cost and the least-cost solution path is found subsequent to refining a reflection wedge. Figure 159 depicts the Map1 problem, the initial solution and the resulting bounding box. Table 23 provides wedge identifications (as did Table 22 for problem 1) and A* evaluations. Figures 160A, 160B and 160C illustrate the wedge search tree for the second demonstration problem while Figures 161 through 178 detail the solution process.

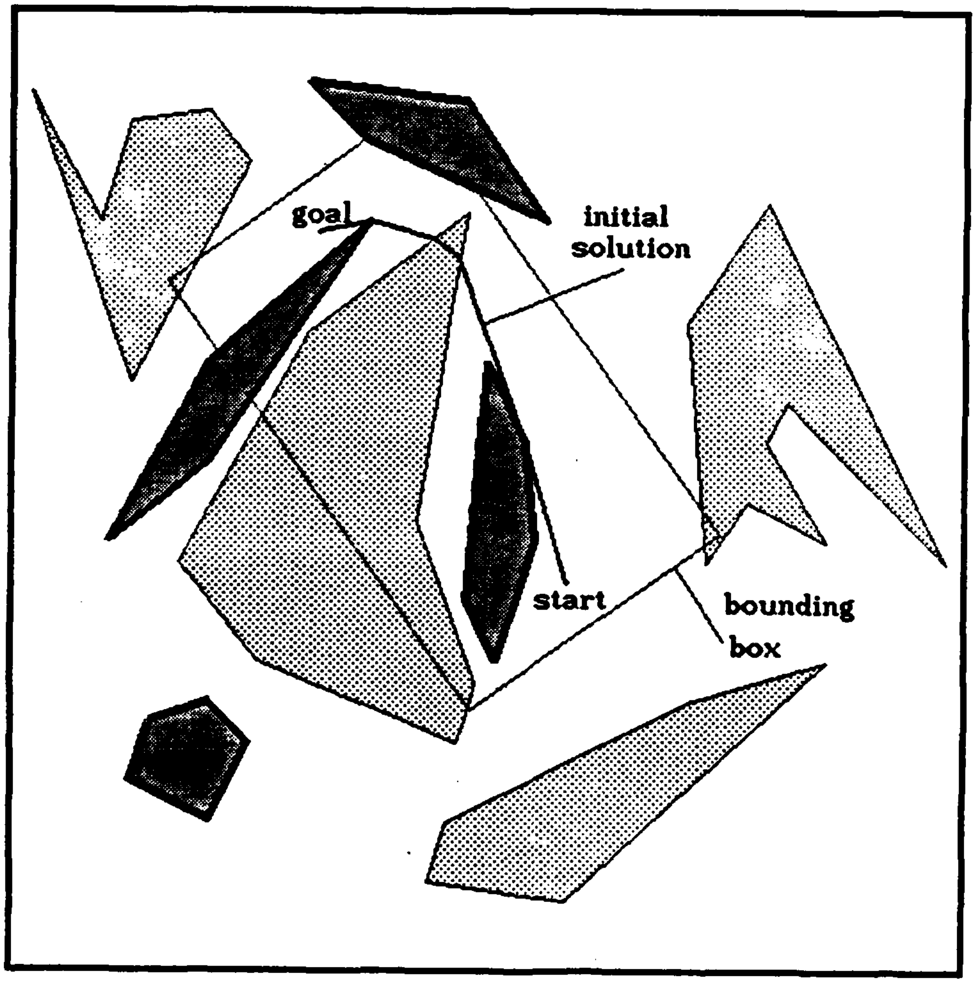


Figure 142. Demonstration Problem 1

TABLE 22

PROBLEM 1 WEDGE DESCRIPTIONS

Wedge	Boundary	A* Evaluations		
ID	Paths	g(W)	h(W)	f(W)
W1	S-a S-z1	0.00	21.51	21.51
W2	S-z1 S-a	0.00	21.51	21.51
W3	S-b S-z1	0.00	21.51	21.51
W4	S-z1 S-c-d-z2	0.00	21.51	21.51
W5	S-c-d-z2 S-c-e-f-g-h	5.32	23.20	28.62
W6	S-z1 S-z3	0.00	21.51	21.51
W7	S-z3 S-c-d-z2	0.00	31.02	31.02
W8	S-i-j-k-z4 S-z1	0.00	21.51	21.51

TABLE 22 (continued)

PROBLEM 1 WEDGE DESCRIPTIONS

Wedge ID	Boundary Paths	A* Evaluations		
		g(W)	h(W)	f(W)
W9	S-i-l-m-n-o S-i-j-k-z4	14.85	7.28	22.13
W10	S-i-j-k-z4 S-p-z5	0.00	26.21	26.21
W11	S-p-q S-z1	0.00	21.51	21.51
W12	S-p-z5 S-p-q	19.16	7.57	26.73
W13	S-z6 S-z1	0.00	21.51	21.51
W14	S-i-l-m-n-o S-i-j-k-z4	7.28	16.70	23.98
W15	S-i-l-r-s-z7 S-i-l-m-n-o	11.75	10.70	22.45
W16	S-i-l-r-s-z7 S-i-l-m-n-o	11.75	10.70	22.45

TABLE 22 (continued)

PROBLEM 1 WEDGE DESCRIPTIONS

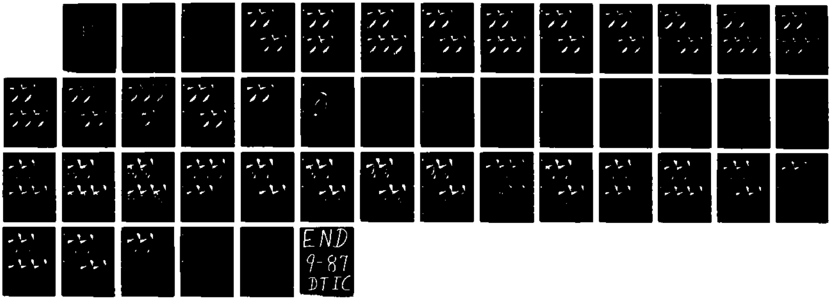
Wedge	Boundary	A* Evaluations		
ID	Paths	g(W)	h(W)	f(W)
W17	S-i-l-r-z8 S-i-l-r-s-z7	23.79	19.96	43.75
W18	S-i-l-r-s-z7 S-i-l-t-z9	11.75	24.07	35.82
W19	S-i-l-t-z10 S-i-l-m-n-o	11.75	10.70	21.45
W20	S-i-l-t-z9 S-i-l-t-z10	20.48	15.34	35.82
W21	S-i-l-t-z10 S-i-l-u-v-w	11.75	10.71	22.76
W22	S-i-l-u-v S-i-l-m-n-o	11.75	13.66	25.41
W23	S-i-l-t-z10 S-i-l-x-z11	11.75	12.64	24.39
W24	S-i-l-x-z11 S-i-l-u-v-w	11.75	21.09	32.84

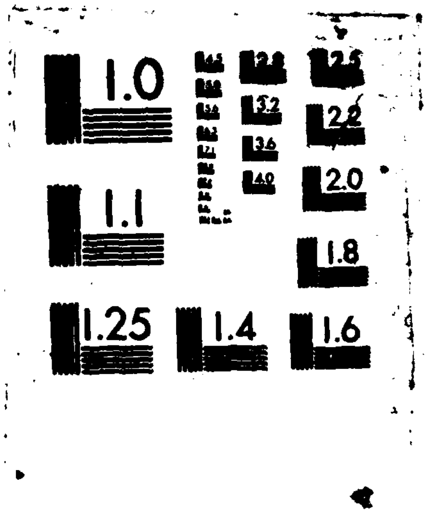
TABLE 22 (continued)

PROBLEM 1 WEDGE DESCRIPTIONS

Wedge	Boundary	A* Evaluations		
ID	Paths	g(W)	h(W)	f(W)
W25	S-i-y-z-aa-z12 S-i-j-k-z4	7.82	17.96	25.24
W26	S-i-y-z-aa-z13 S-i-y-z-aa-z12	23.82	2.55	26.37
W27	S-i-l-ab-ac-v-z16 S-i-l-ab-ac-v-ad	27.16	5.02	32.18
W28	S-i-y-z-aa-z13 S-i-y-z-aa-z12	23.82	2.54	26.36
RW1	S-i-l-ab-ac-z15 S-i-l-ab-ac-v-z14	23.93	3.71	27.64

AD-A183 900 SOLVING A CLASS OF SPATIAL REASONING PROBLEMS: 5/5
MINIMAL-COST PATH PLANNING IN THE CARTESIAN PLANE(U)
NAVAL POSTGRADUATE SCHOOL MONTEREY CA R F RICHBOURG
UNCLASSIFIED JUN 87 F/G 12/9 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

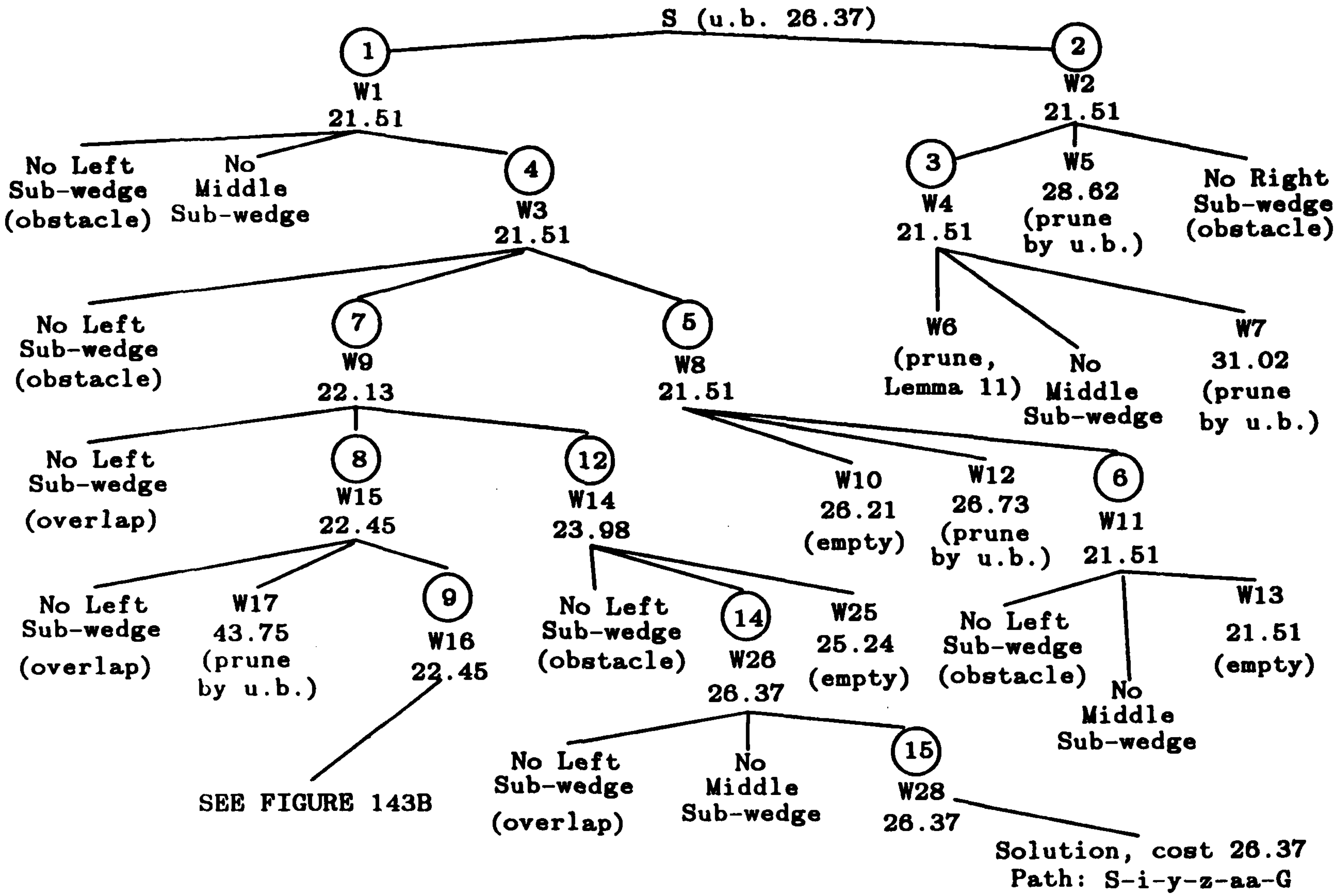


Figure 143A. Problem 1 Wedge Tree

(Continued from Figure 143A)

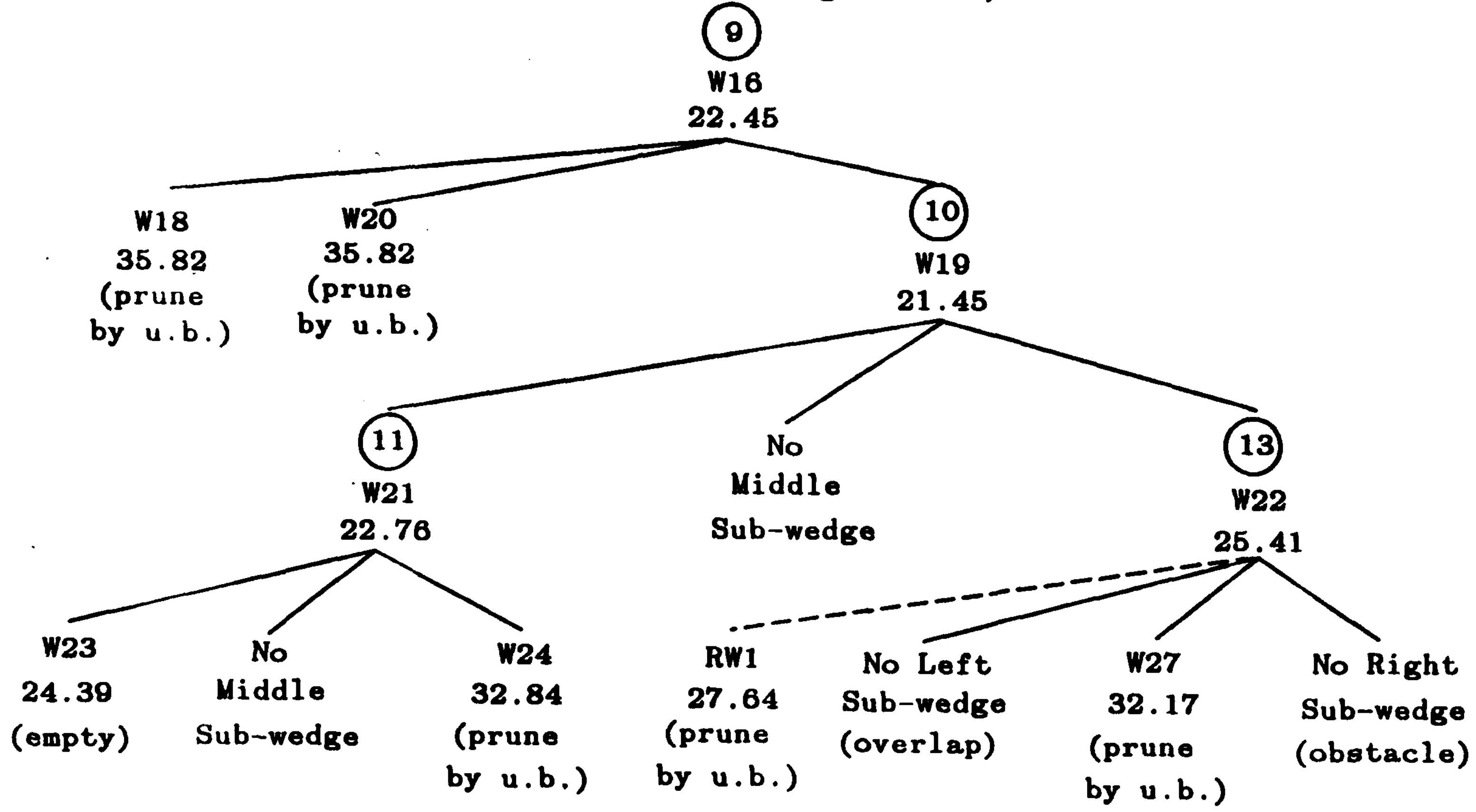
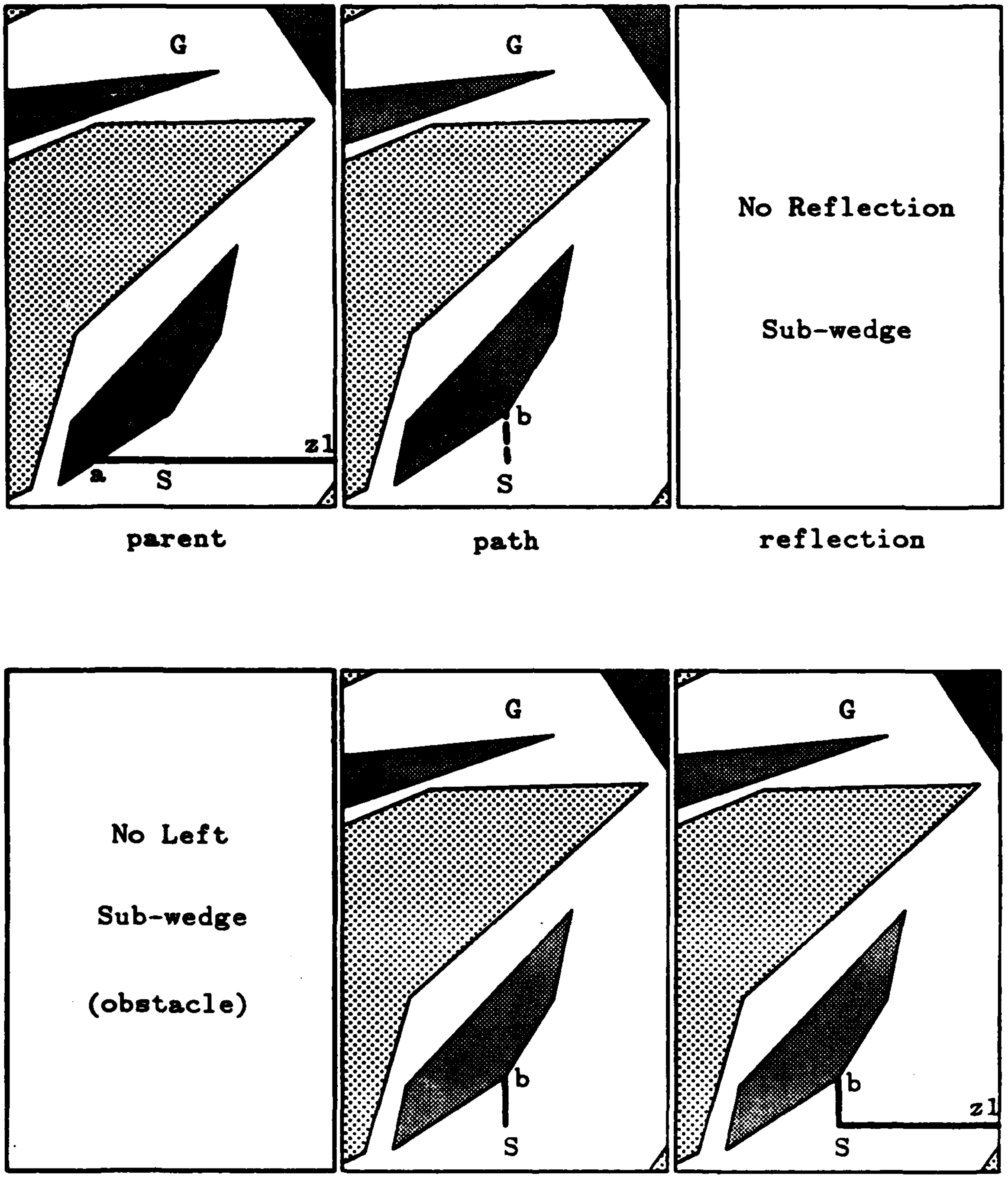


Figure 143B. Problem 1 Wedge Tree (continued)



parent

path

reflection

left

middle

right

Figure 144. Refinement of Wedge W1

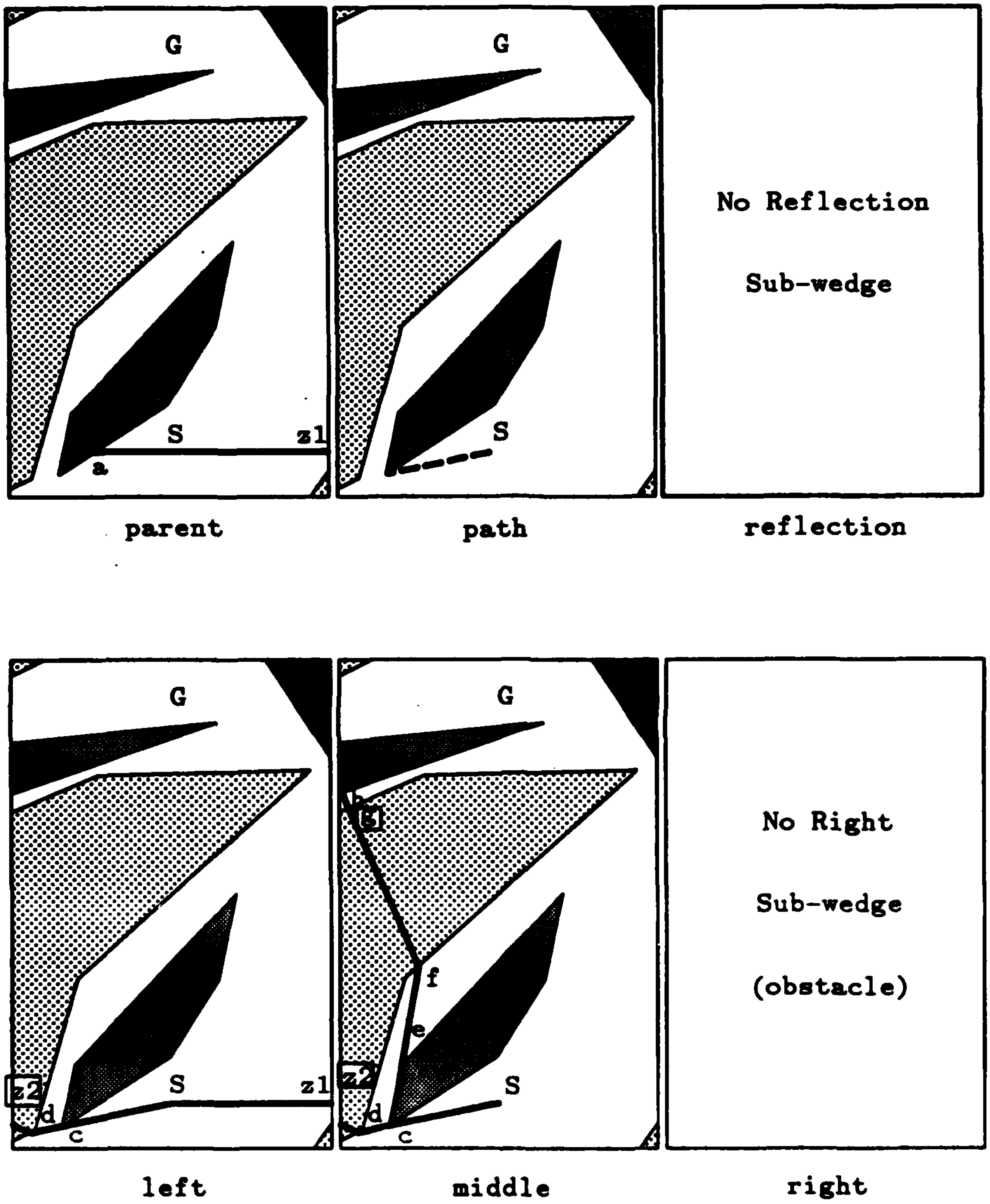


Figure 145. Refinement of Wedge W2

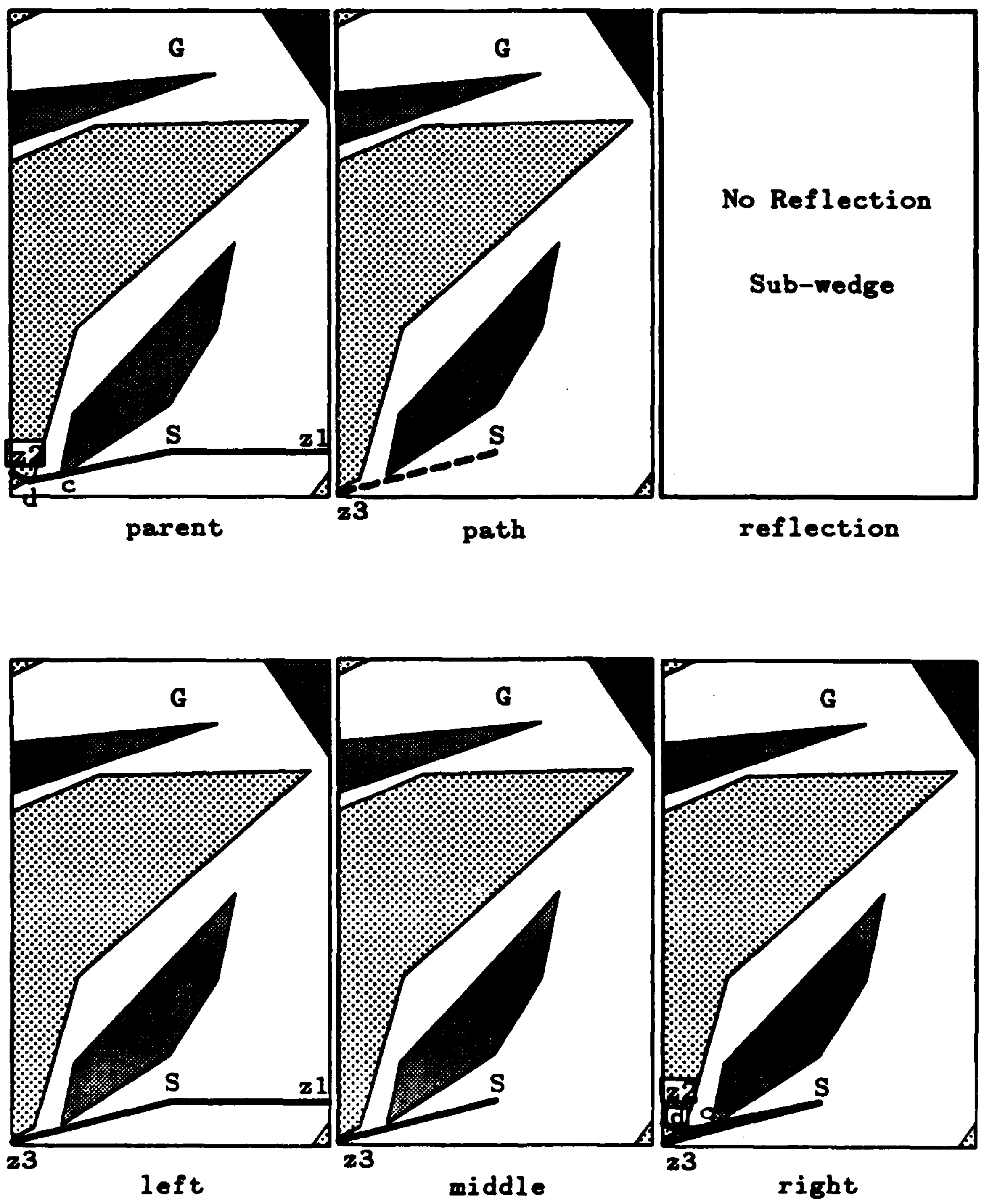
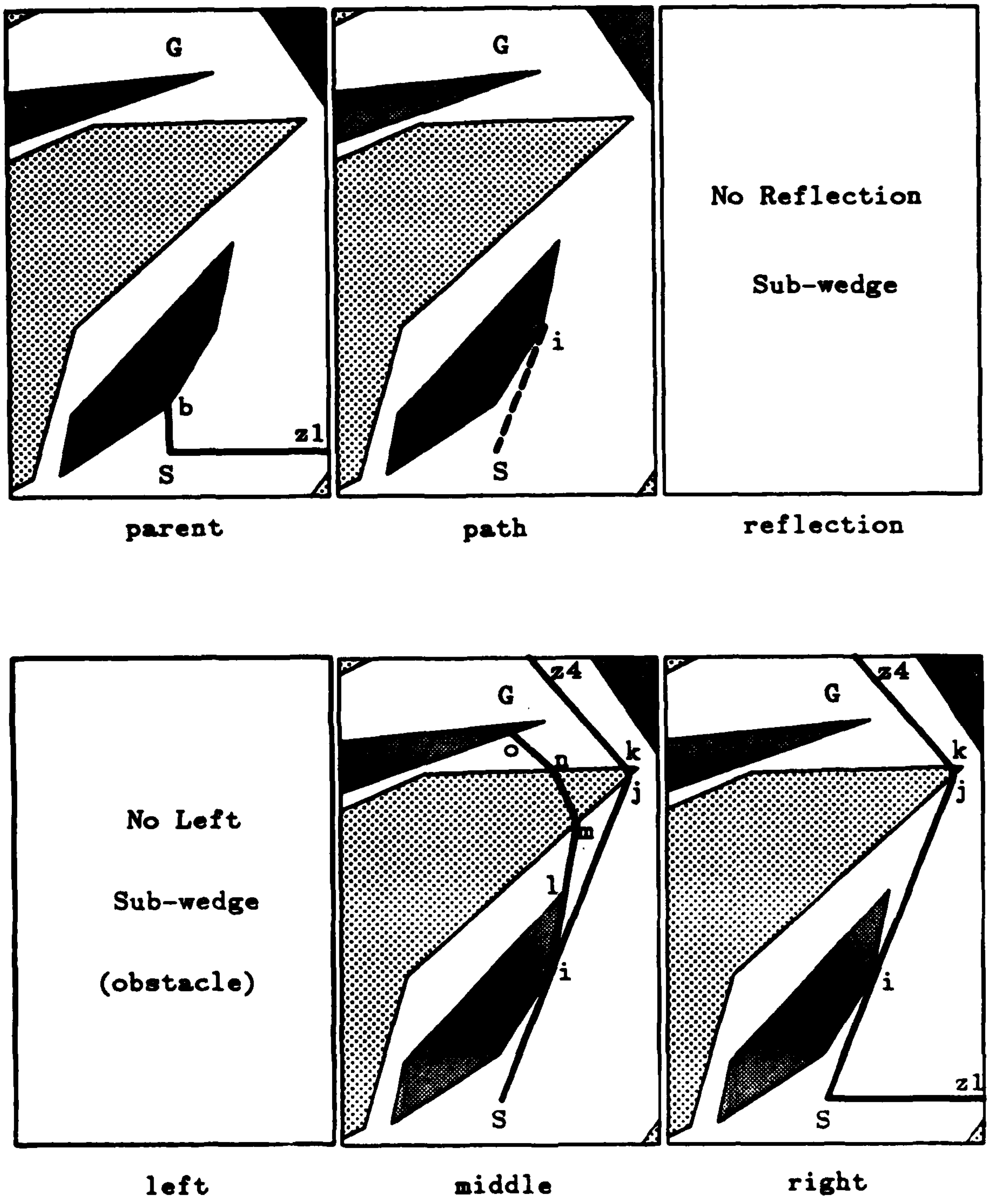


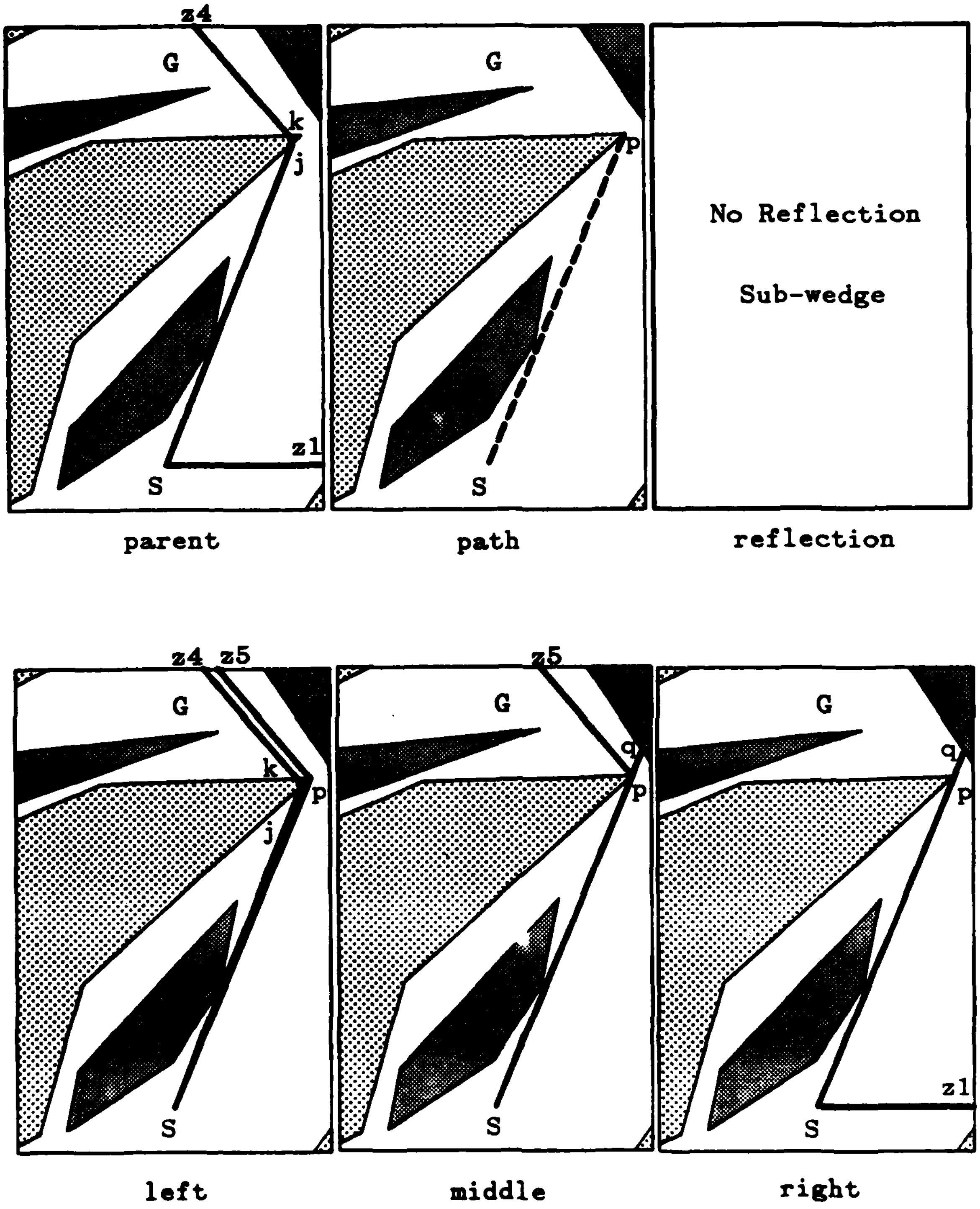
Figure 146. Refinement of Wedge W4



parent path reflection

left middle right

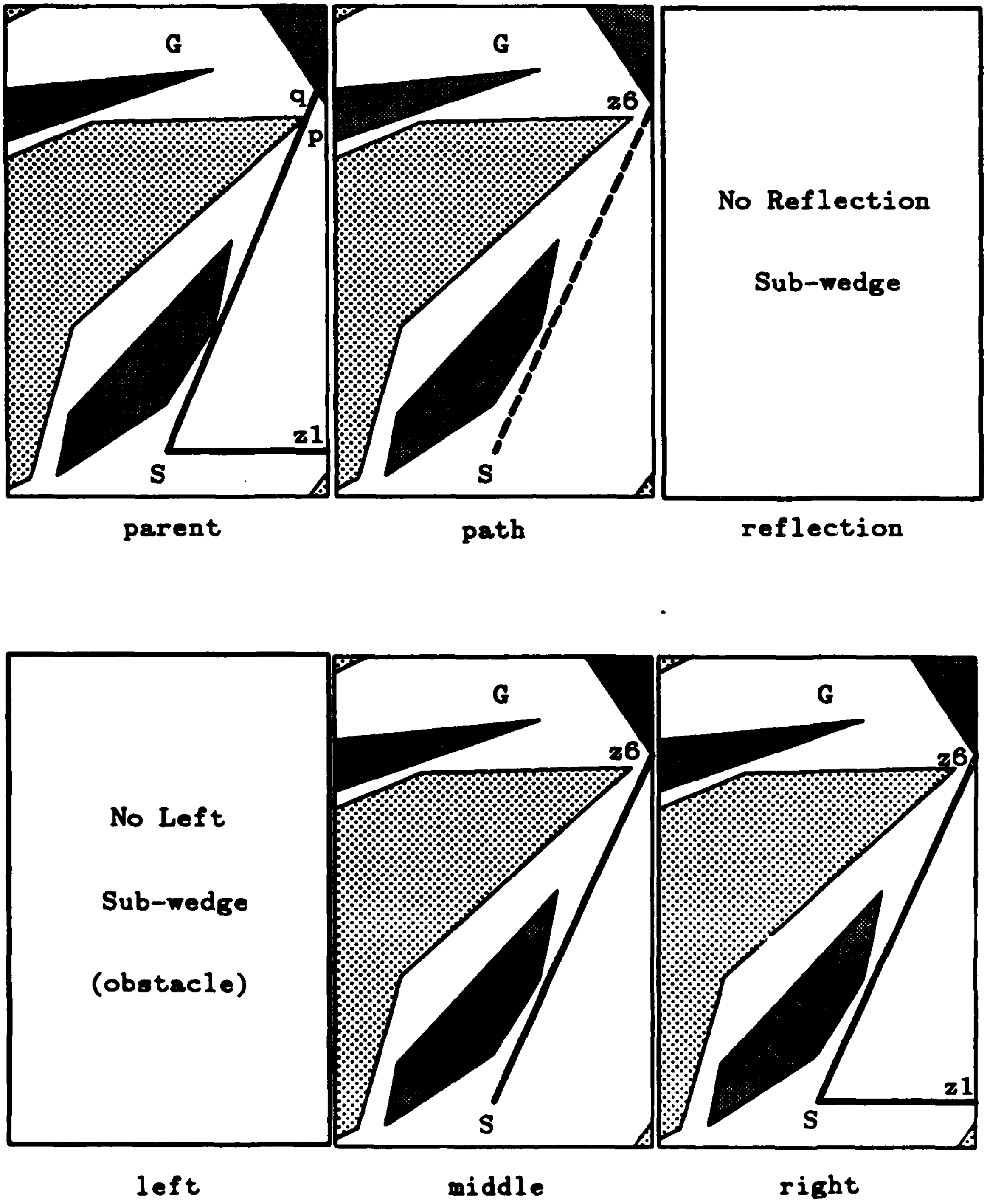
Figure 147. Refinement of Wedge W3



parent path reflection

left middle right

Figure 148. Refinement of Wedge W8



parent

path

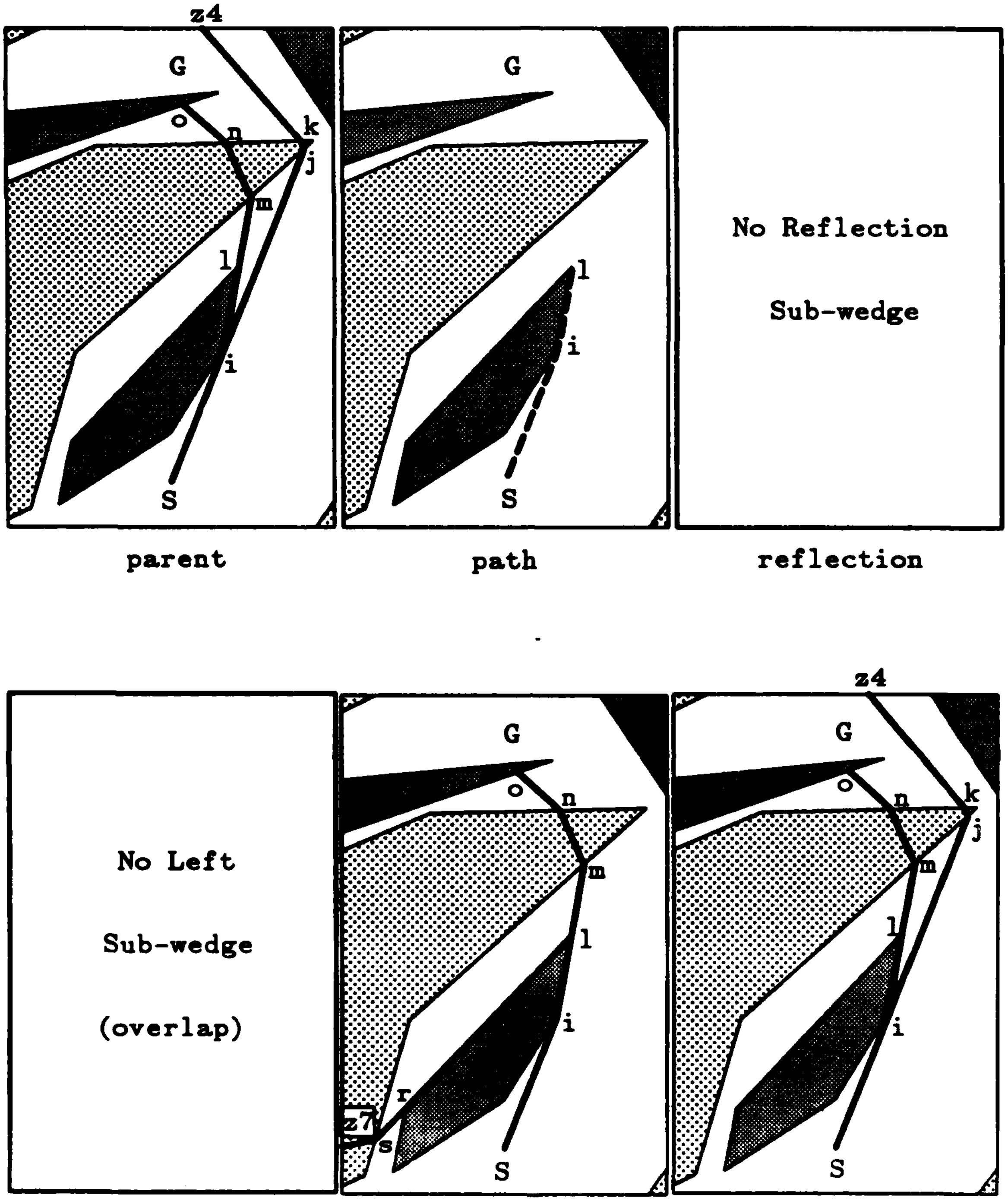
reflection

left

middle

right

Figure 149. Refinement of Wedge W11



parent path reflection

left middle right

Figure 150. Refinement of Wedge W9

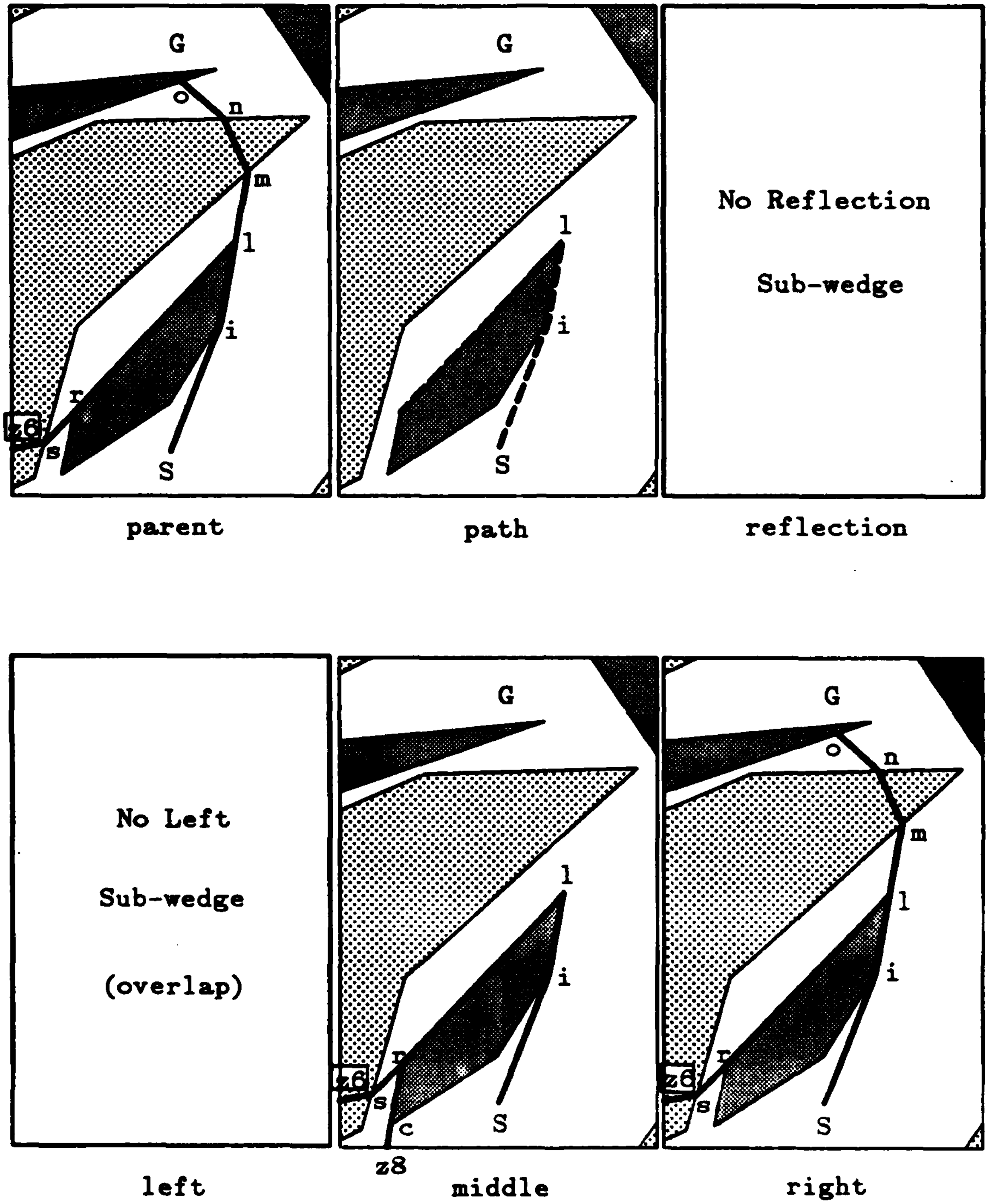
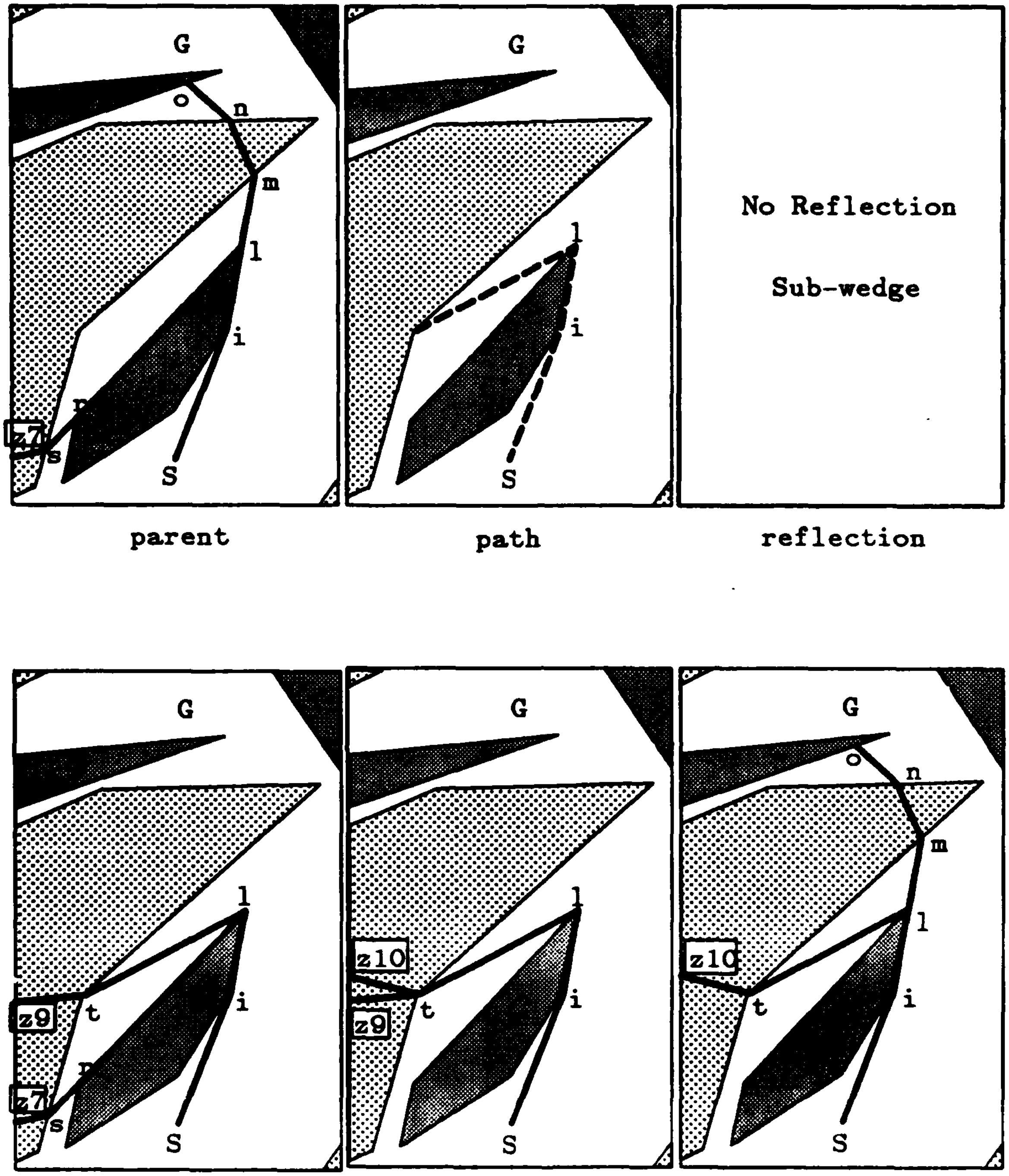


Figure 151. Refinement of Wedge W15



parent

path

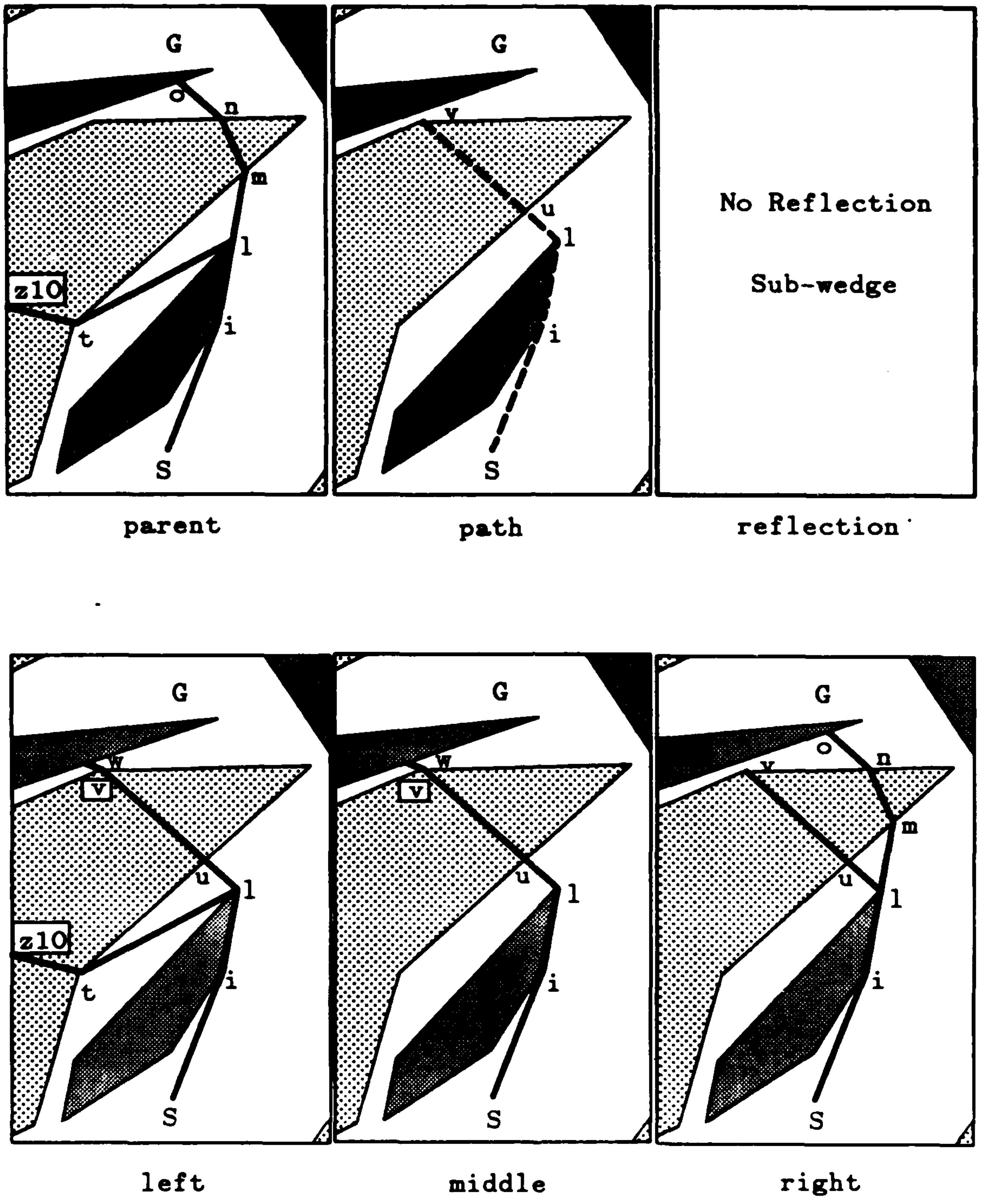
reflection

left

middle

right

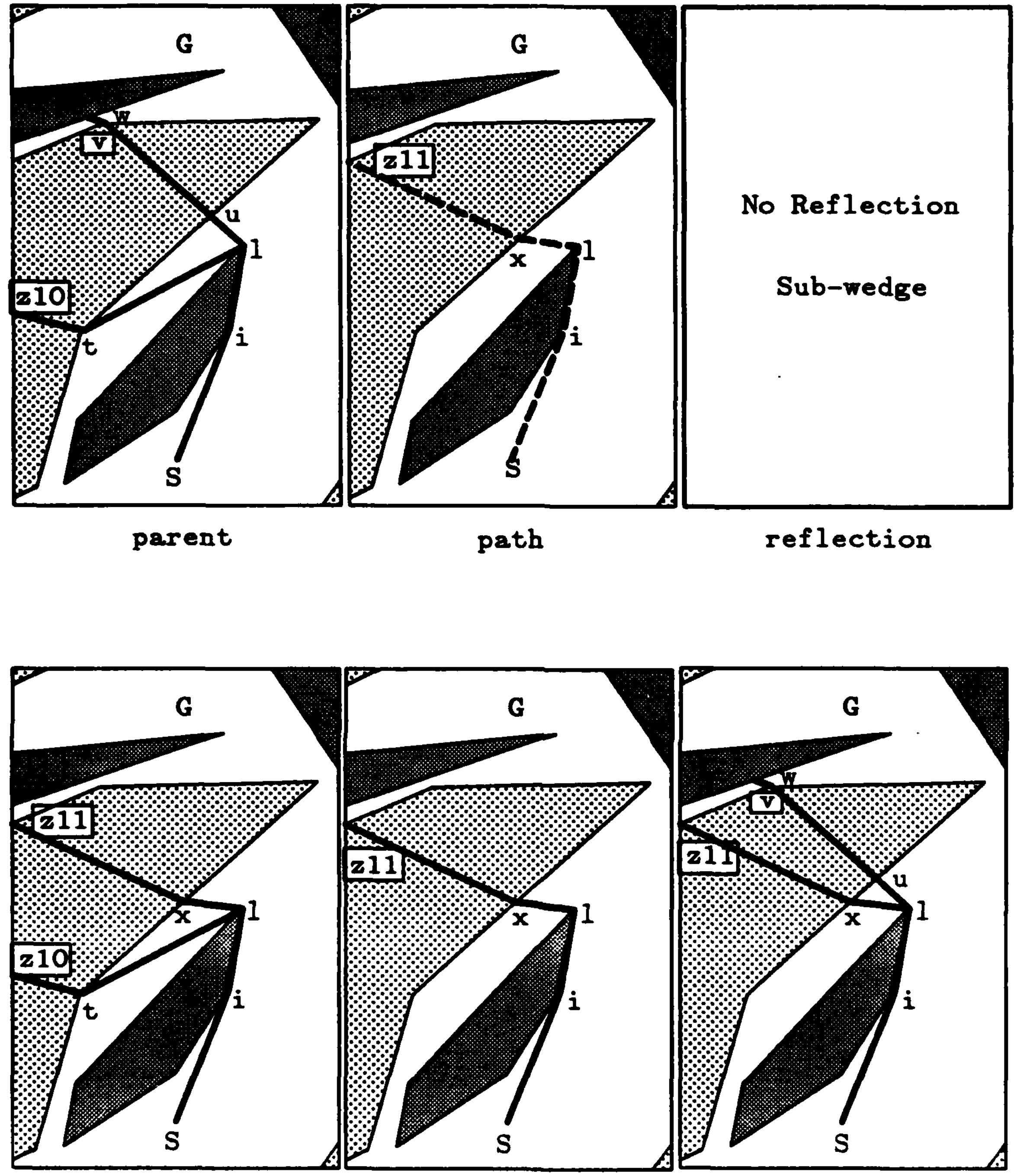
Figure 152. Refinement of Wedge W16



parent path reflection

left middle right

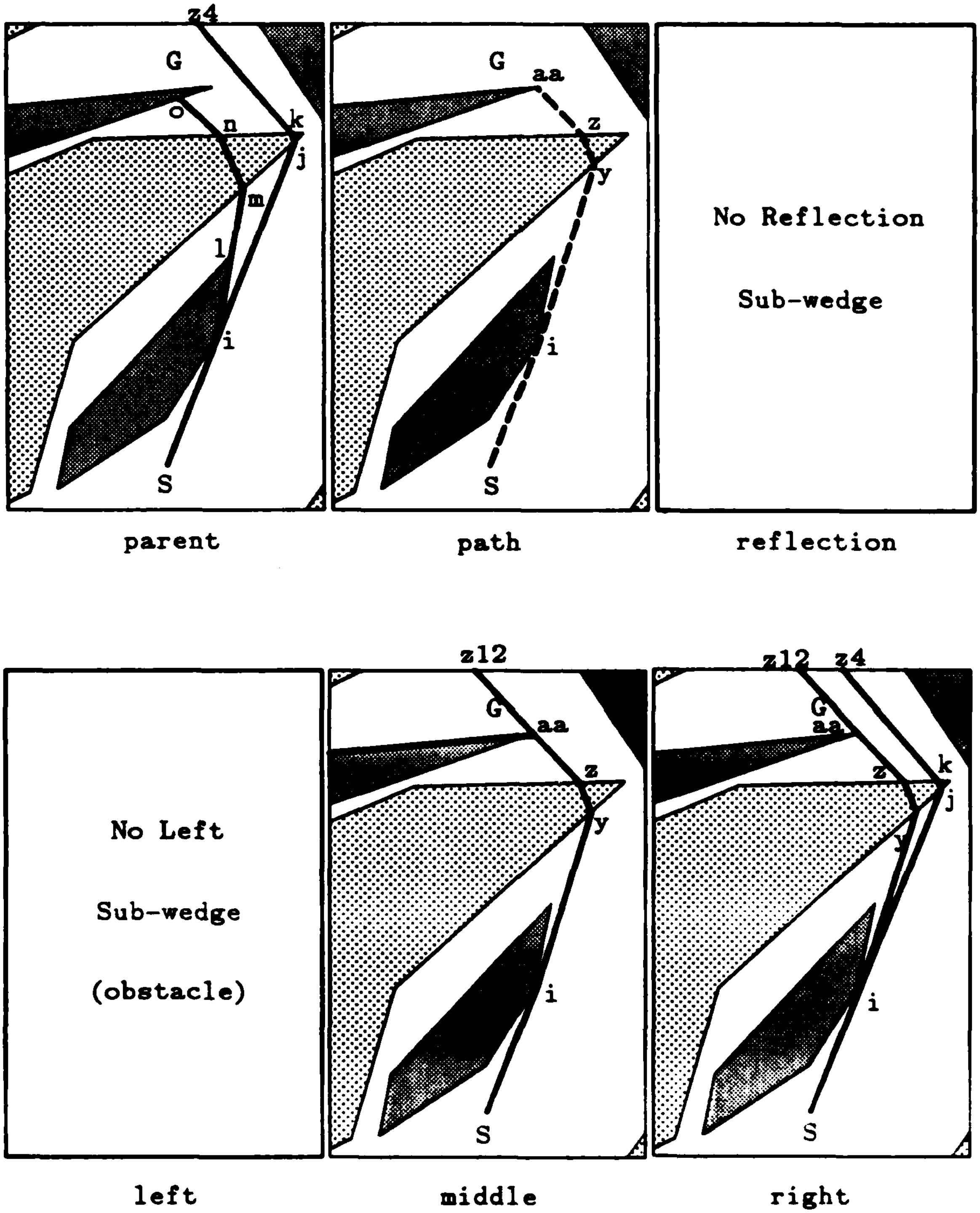
Figure 153. Refinement of Wedge W19



parent path reflection

left middle right

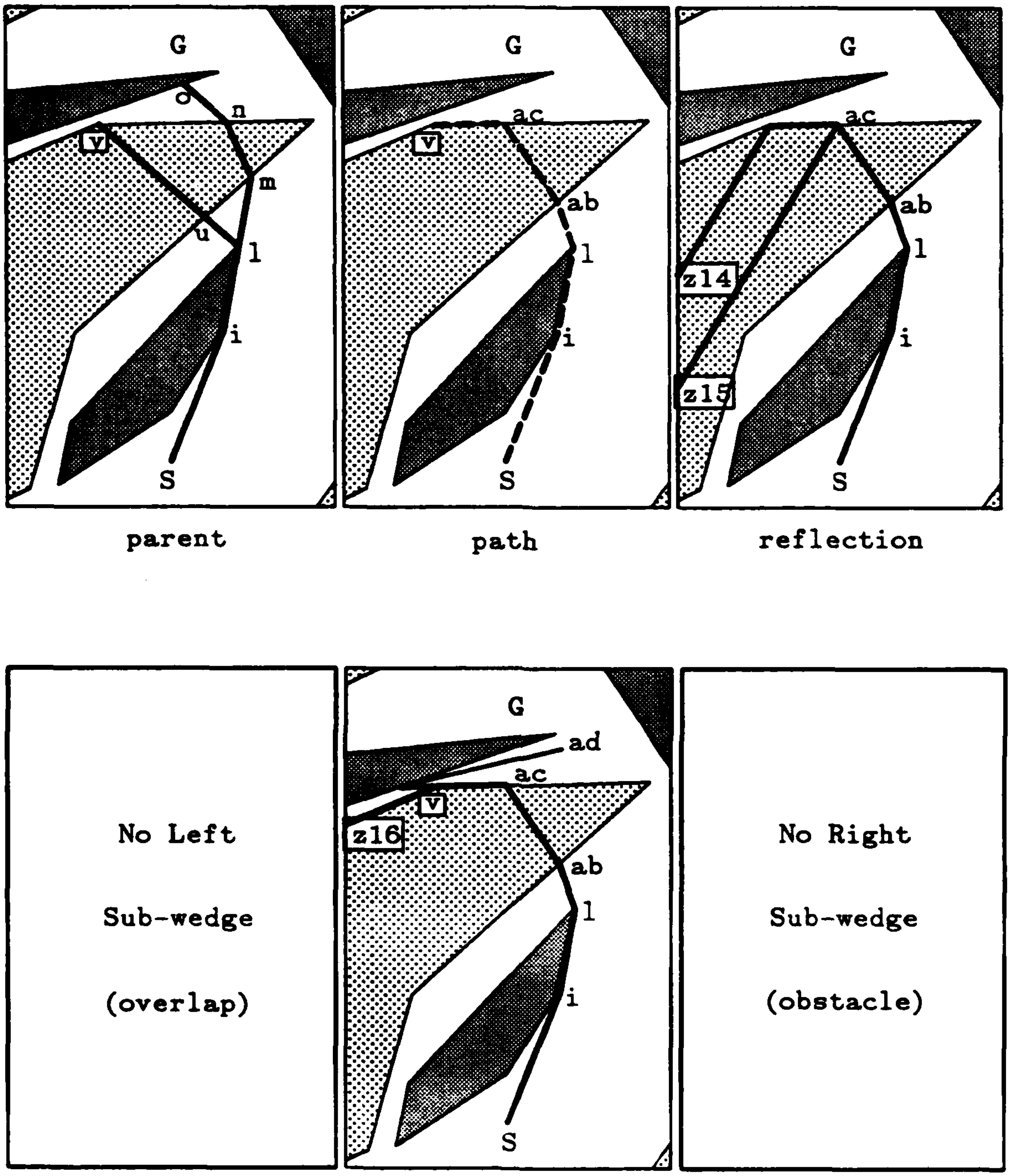
Figure 154. Refinement of Wedge W21



parent path reflection

left middle right

Figure 155. Refinement of Wedge W14



parent

path

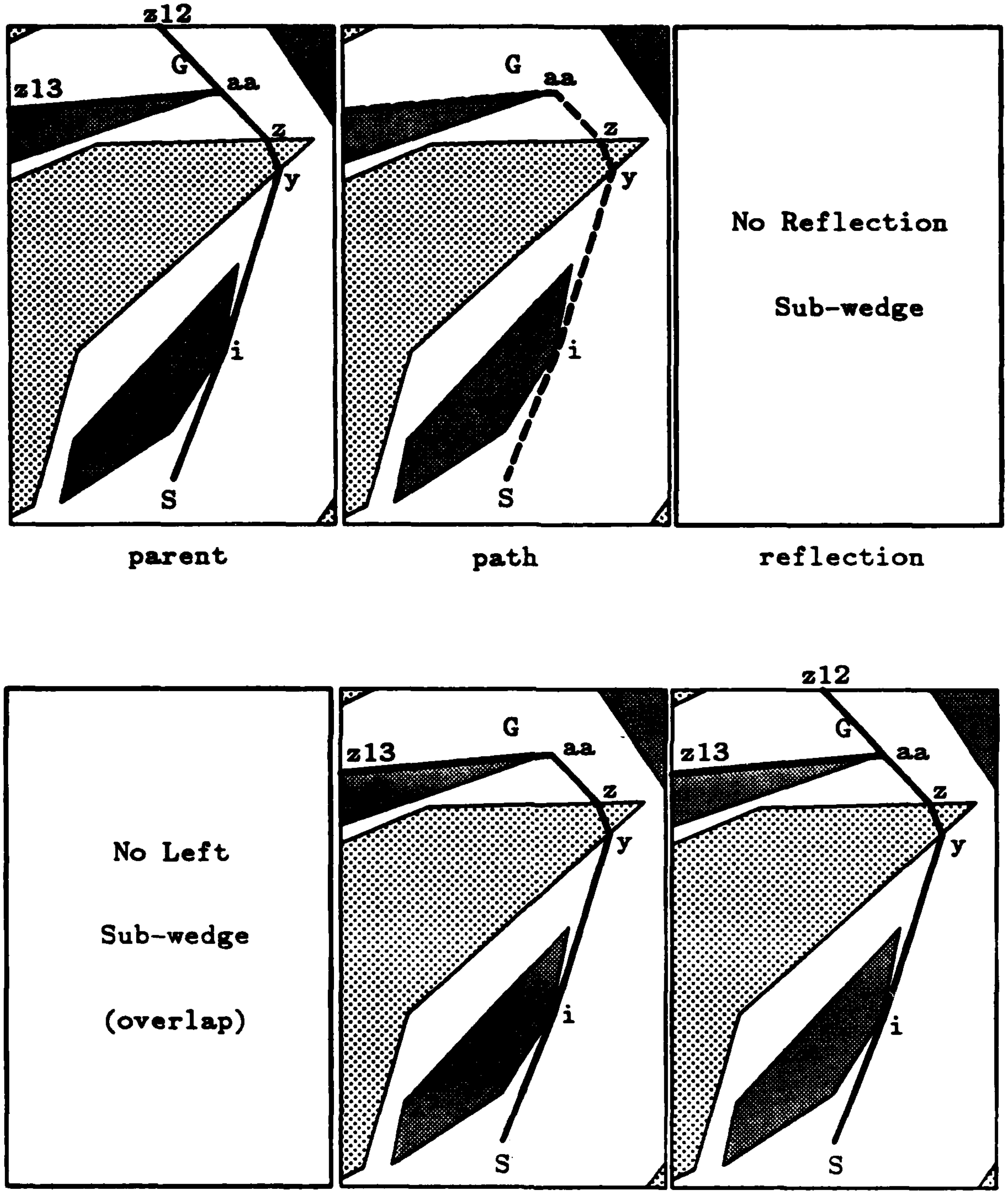
reflection

left

middle

right

Figure 156. Refinement of Wedge W22



parent

path

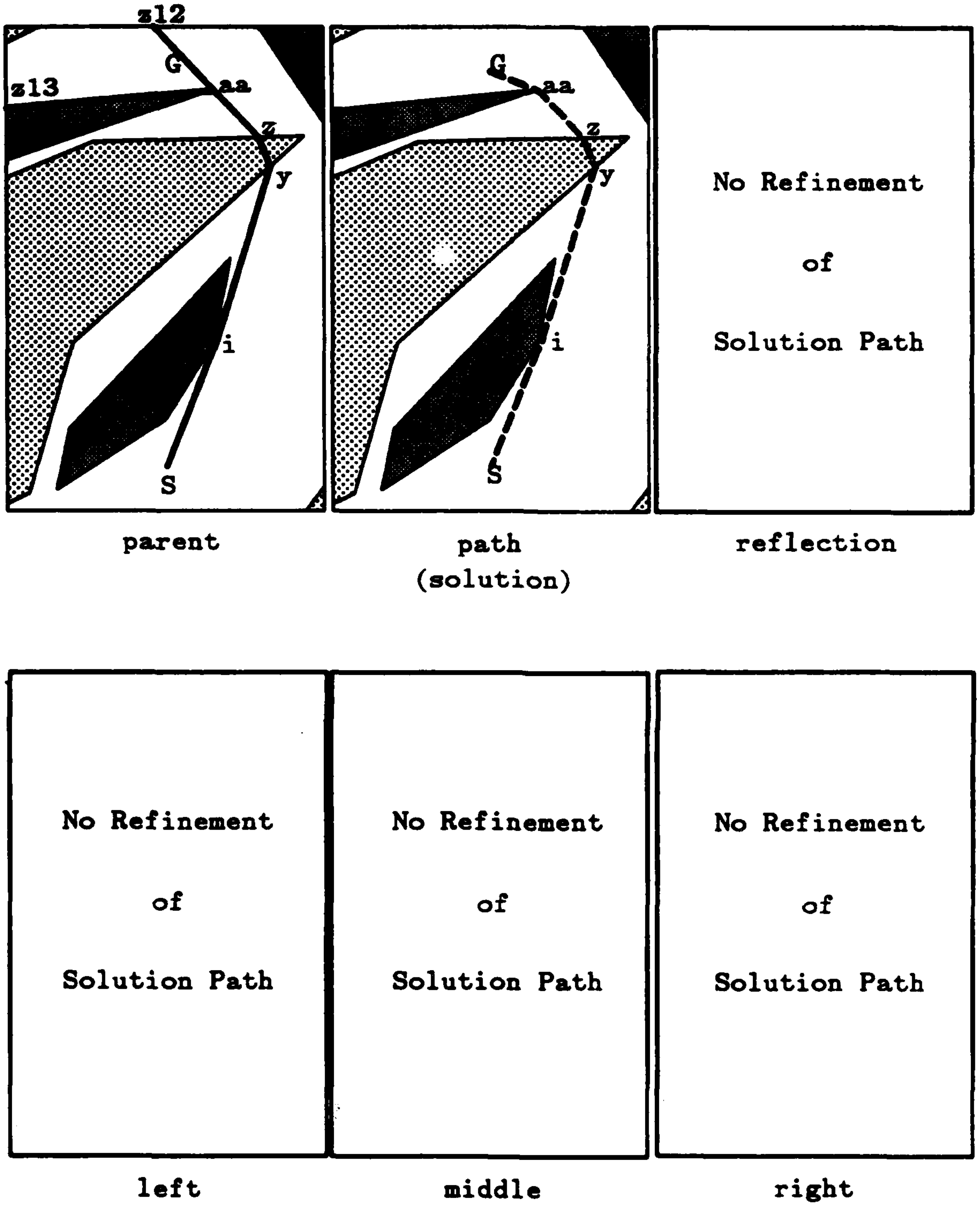
reflection

left

middle

right

Figure 157. Refinement of Wedge W26



parent

path
(solution)

reflection

left

middle

right

Figure 158. Solution Path Through Wedge W28

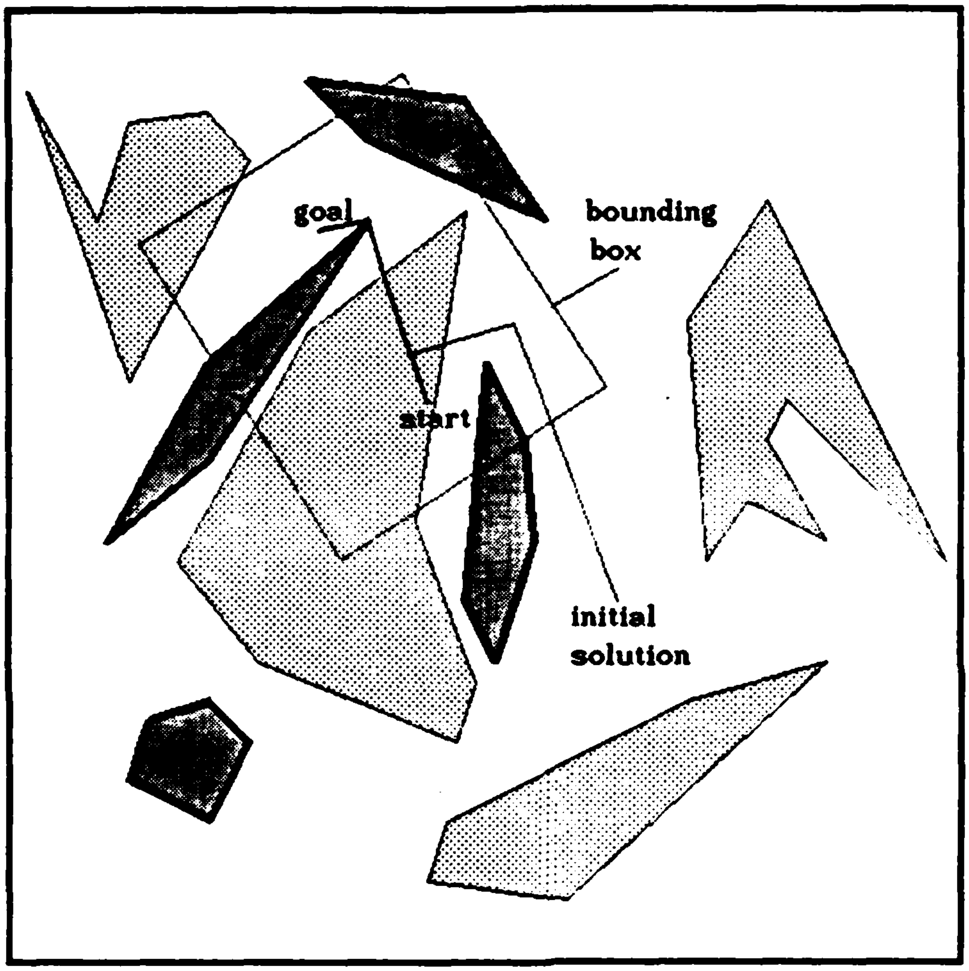


Figure 159. Demonstration Problem 2

TABLE 23

PROBLEM 2 WEDGE DESCRIPTIONS

Wedge ID	Boundary Paths	A* Evaluations		
		g(W)	h(W)	f(W)
W1	S-z1 S-a-b	0.00	10.12	10.12
W2	S-a-b S-z1	0.00	10.12	10.12
W3	S-z1 S-d-b-z3	0.00	10.12	10.12
W4	S-c-b-g S-c-b-h	10.50	7.56	18.06
W5	S-a-b S-z4	0.00	11.07	11.07
W6	S-z4 S-z1	0.00	10.12	10.12
W7	S-z1 S-g-i	0.00	10.12	10.12
W8	S-g-j S-d-b-z3	0.00	16.5	16.5

TABLE 23 (continued)

PROBLEM 2 WEDGE DESCRIPTIONS

Wedge	Boundary	A* Evaluations		
ID	Paths	g(W)	h(W)	f(W)
W9	S-z1 S-z5	0.00	10.12	10.12
W10	S-z5 S-g-i	0.00	18.91	18.91
W11	S-c-b-h S-k-z4	1.06	10.63	11.69
W12	S-l-z7 S-k-z4	0.00	11.17	11.17
W13	S-m-n-o S-d-b-z3	0.00	16.99	16.99
W14	S-m-n-z8 S-m-n-o	16.14	2.55	18.69
W15	S-p-q S-d-b-z3	0.00	17.12	17.12
W16	S-r-z9 S-d-b-z3	0.00	19.49	19.49

TABLE 23 (continued)

PROBLEM 2 WEDGE DESCRIPTIONS

Wedge	Boundary	A* Evaluations		
ID	Paths	g(W)	h(W)	f(W)
W17	S-c-b-g-z5 S-c-b-g-u	20.49	5.02	25.52
W18	S-c-b-g-u S-c-b-h	10.50	7.56	18.06
W19	S-c-b-n-z8 S-c-b-n-v-z11	15.52	2.55	18.07
W20	S-c-b-n-v-z11 S-c-b-h	10.59	7.47	18.06
W21	S-c-b-n-v-z11 S-c-b-w-z12	10.59	7.47	18.06
W22	S-c-b-w-z12 S-c-b-w-z13	16.60	5.15	21.75
W23	S-c-b-n-v-z11 S-c-b-z14	10.59	16.27	26.86
W24	S-c-b-z14 S-c-b-z12	10.59	7.47	18.06

TABLE 23 (continued)

PROBLEM 2 WEDGE DESCRIPTIONS

Wedge	Boundary	A* Evaluations		
ID	Paths	g(W)	h(W)	f(W)
W25	S-c-b-n-z8 S-c-b-n-v-z11	15.52	2.55	18.07
W26	S-c-x-y-n-z8 S-c-x-y-n-z15	14.84	2.55	17.39
W27	S-c-x-y-n-z8 S-c-x-y-n-z15	14.84	2.55	17.39
RW1	S-c-e-f S-c-b-z2	1.07	10.25	11.32
RW2	S-k-z4 S-z6	1.06	10.26	11.32
RW3	S-c-b-t S-c-b-g-z10	10.50	7.56	18.06
RW4	S-c-e-f S-c-x-y-n-z8	1.07	10.24	11.31
RW5	S-c-x-y-n-z15 S-c-b-z2	1.07	14.69	15.76

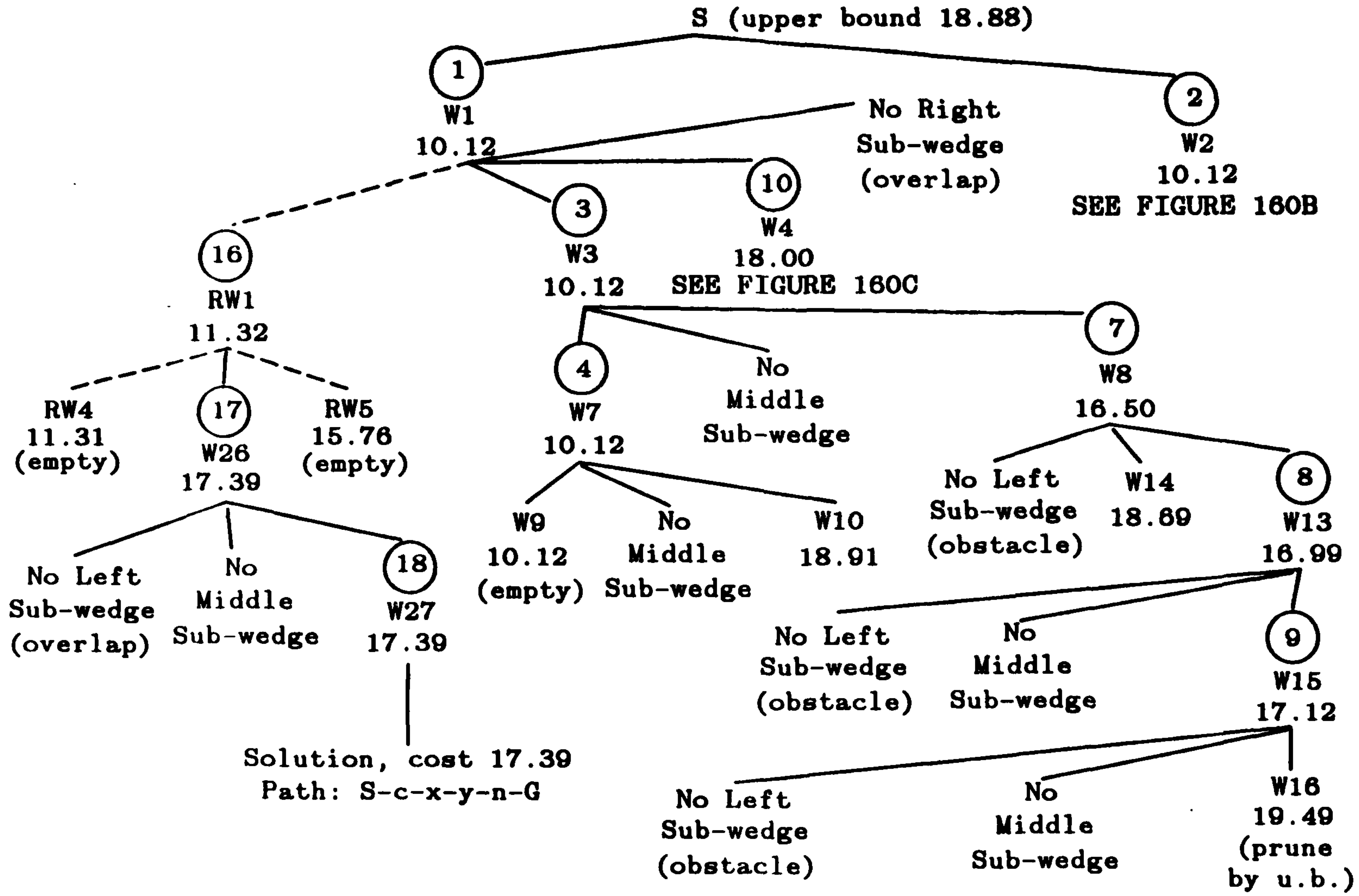


Figure 160A. Problem 2 Wedge Tree

(continued from Figure 160A)

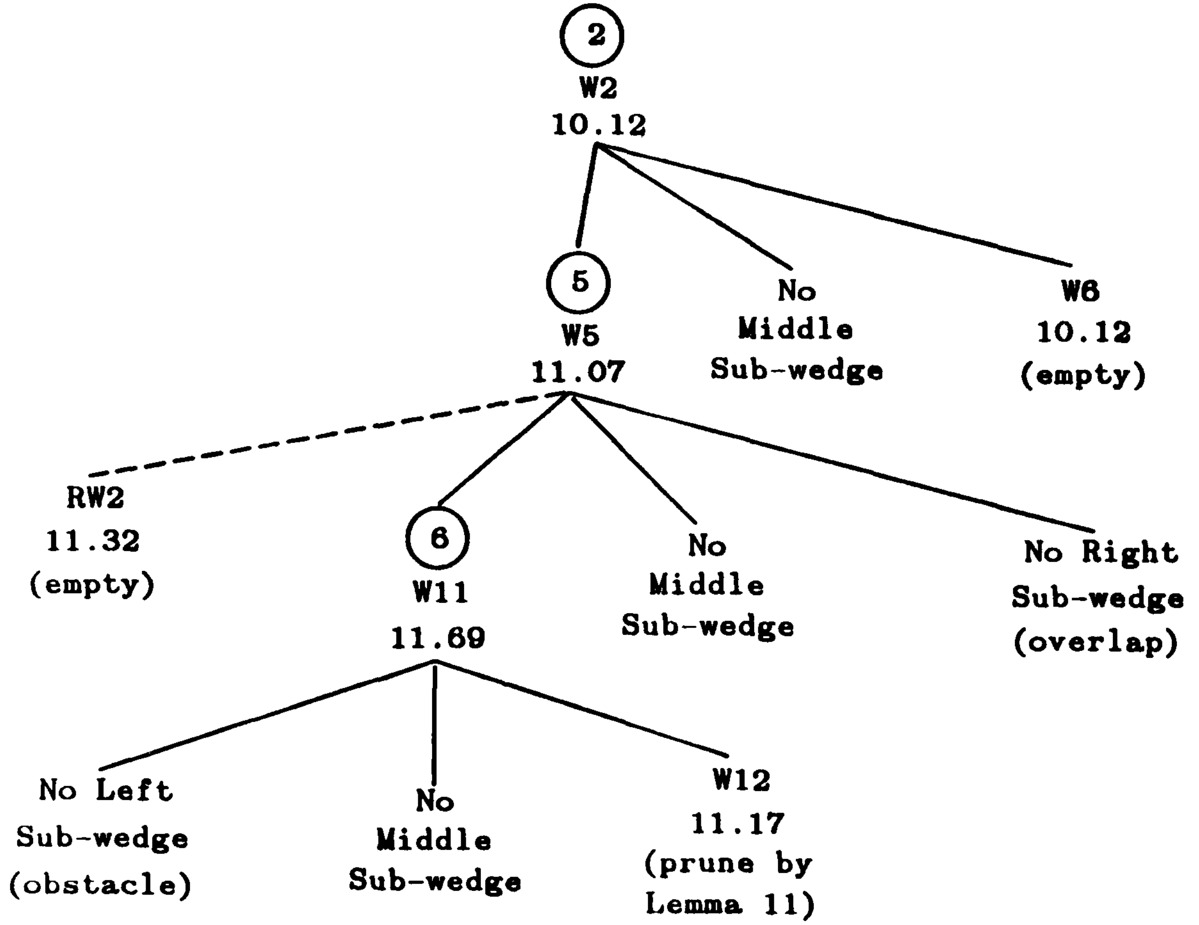


Figure 160B. Problem 2 Wedge Tree (continued)

(continued from Figure 160A)

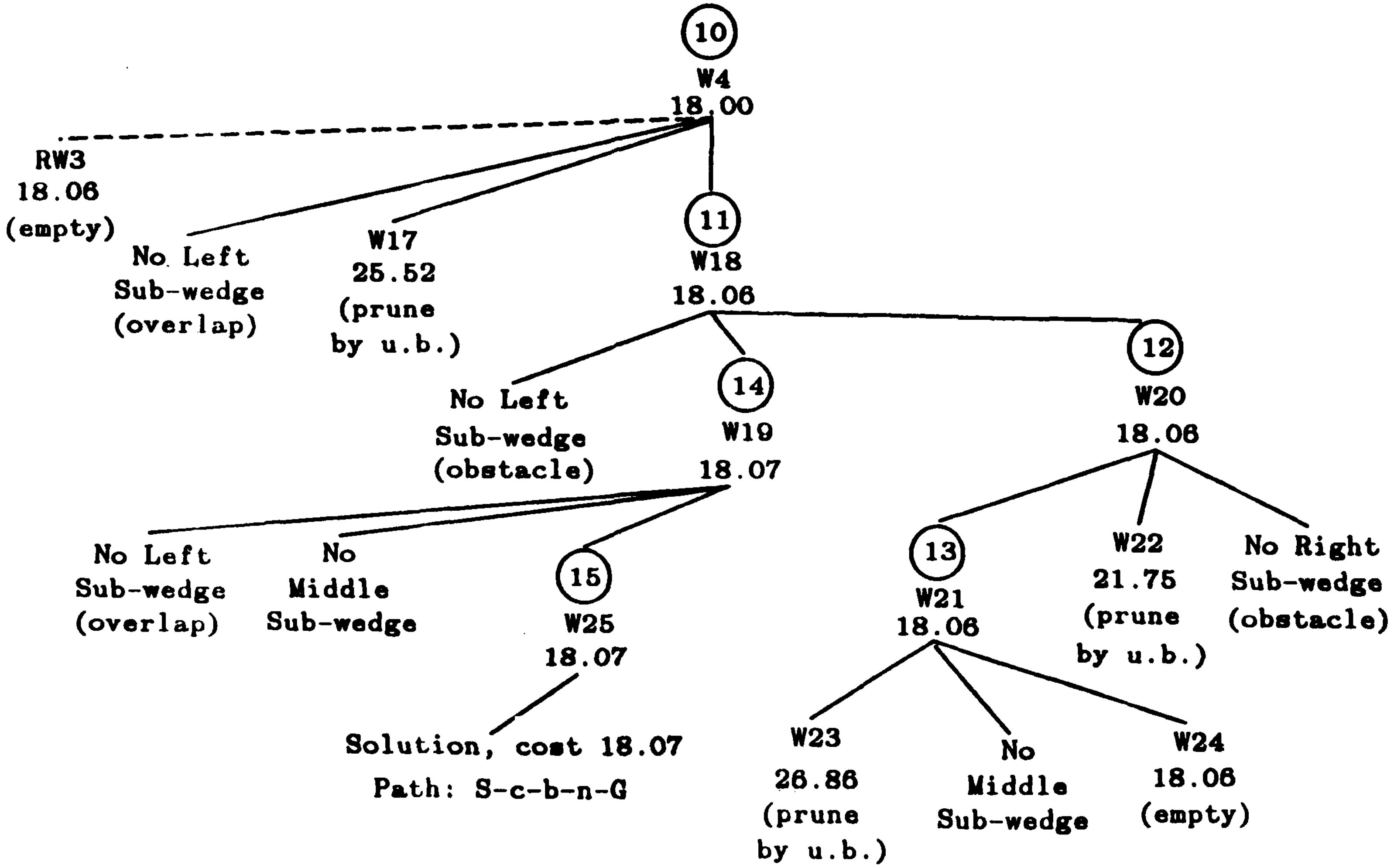


Figure 160C. Problem 2 Wedge Tree (continued)

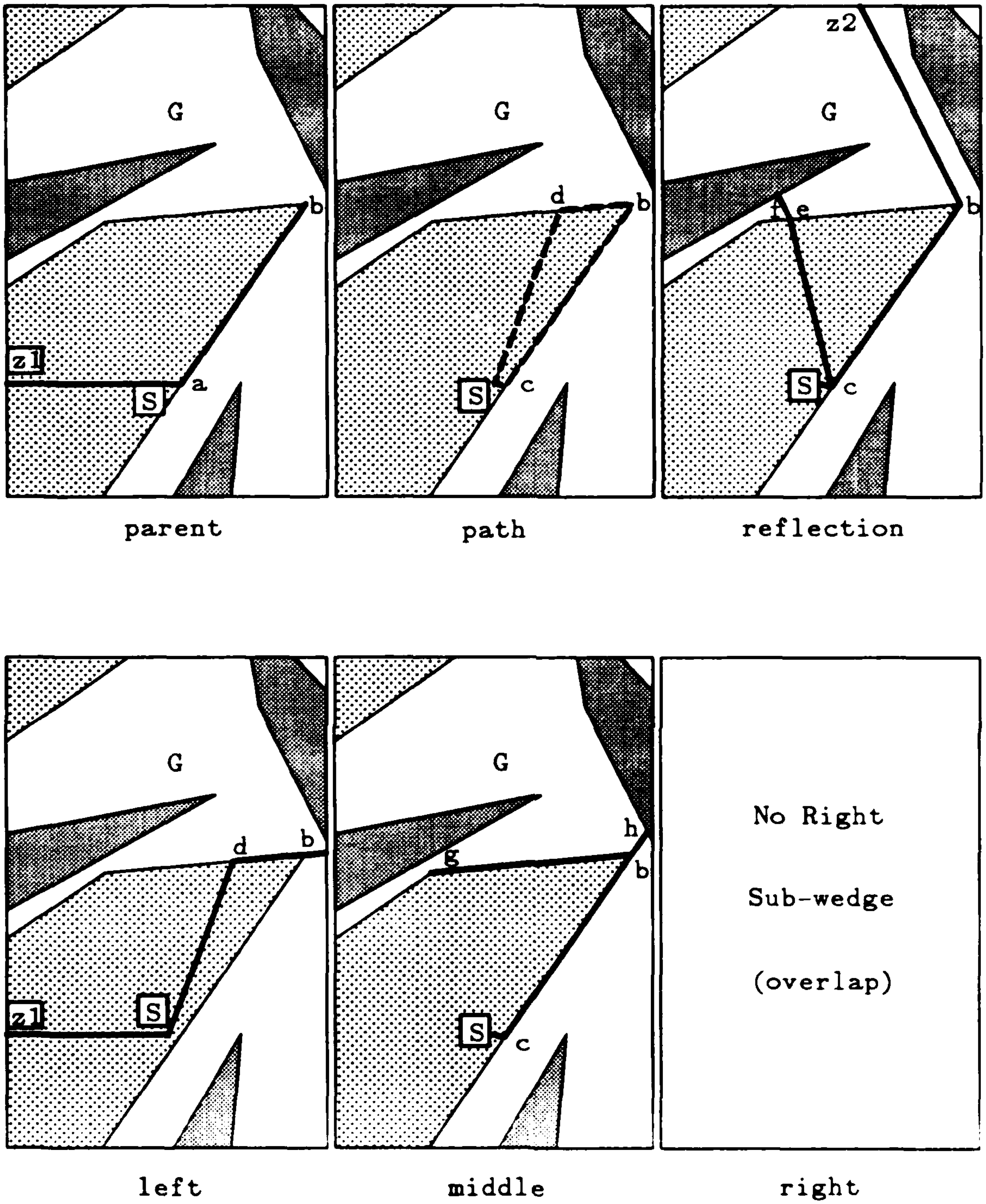
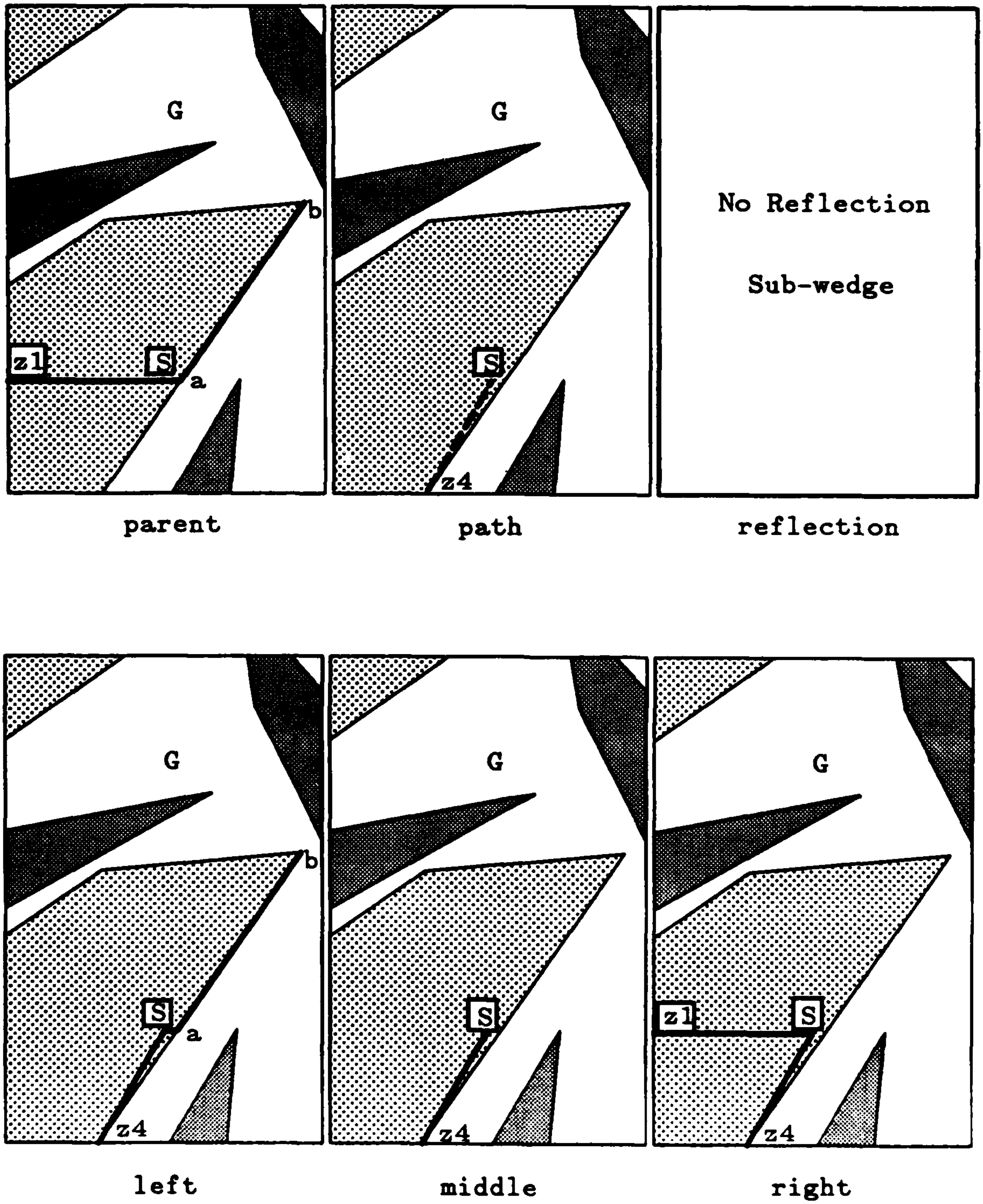


Figure 161. Refinement of Wedge W1



parent

path

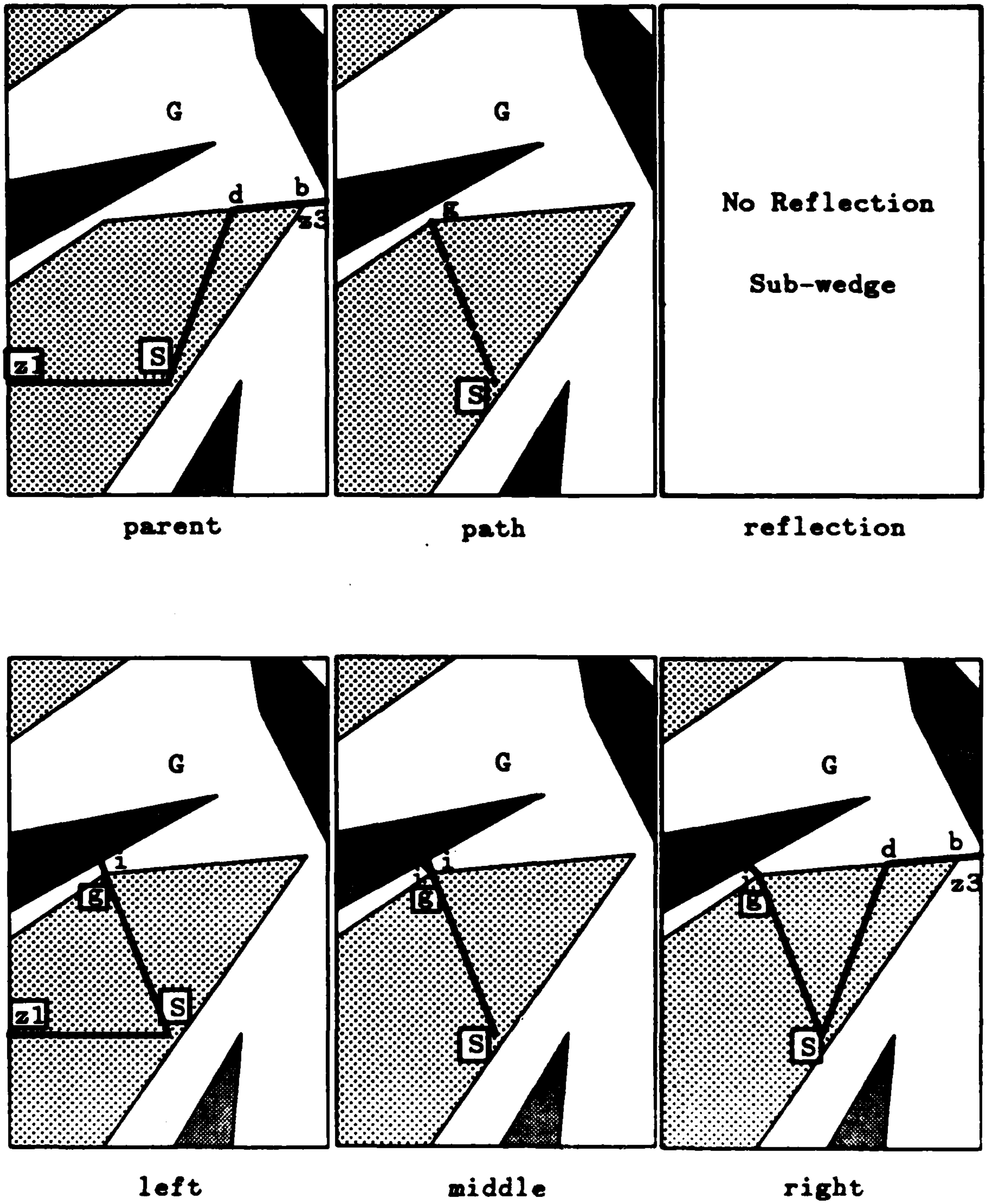
reflection

left

middle

right

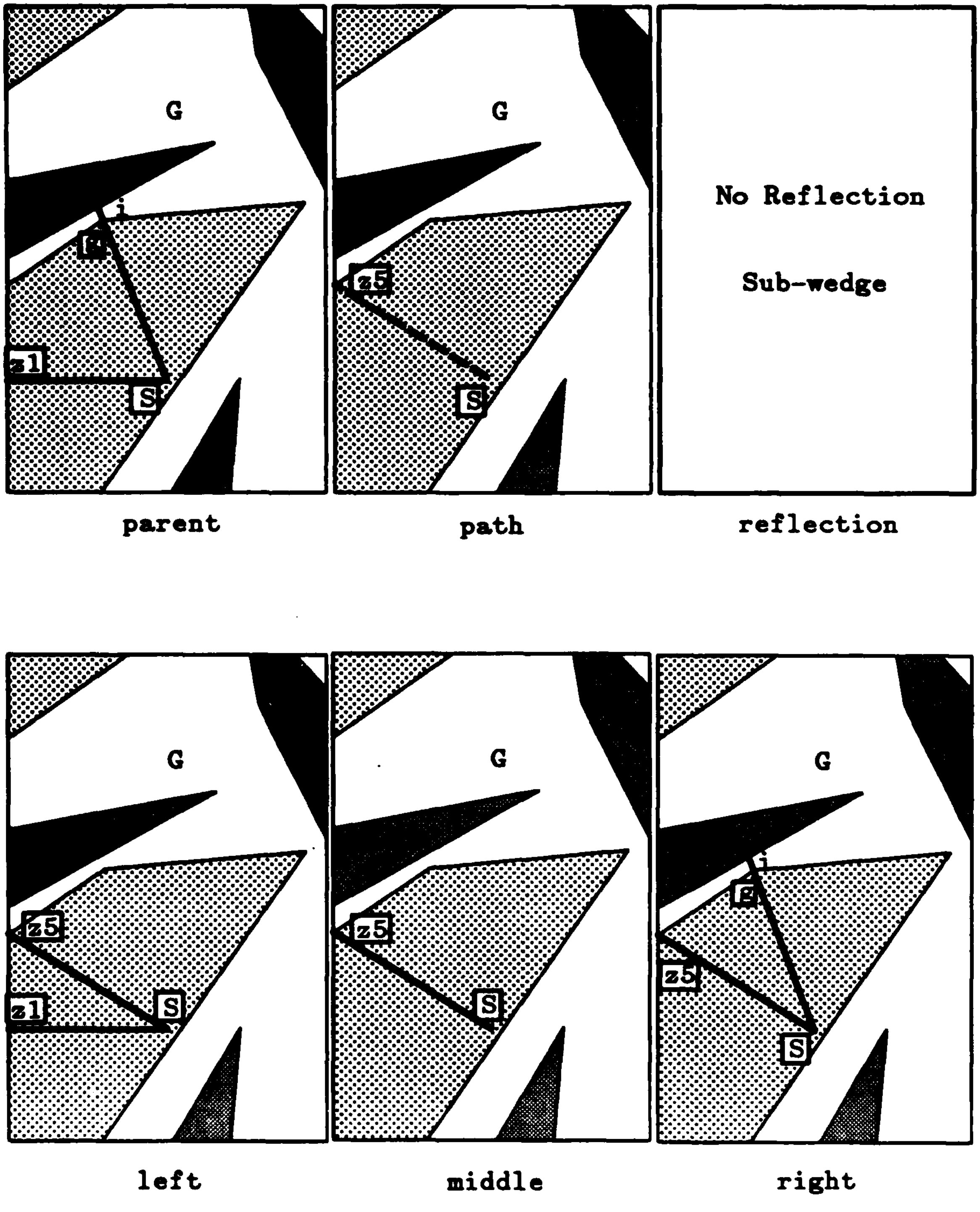
Figure 162. Refinement of Wedge W2



parent path reflection

left middle right

Figure 163. Refinement of Wedge W3



parent

path

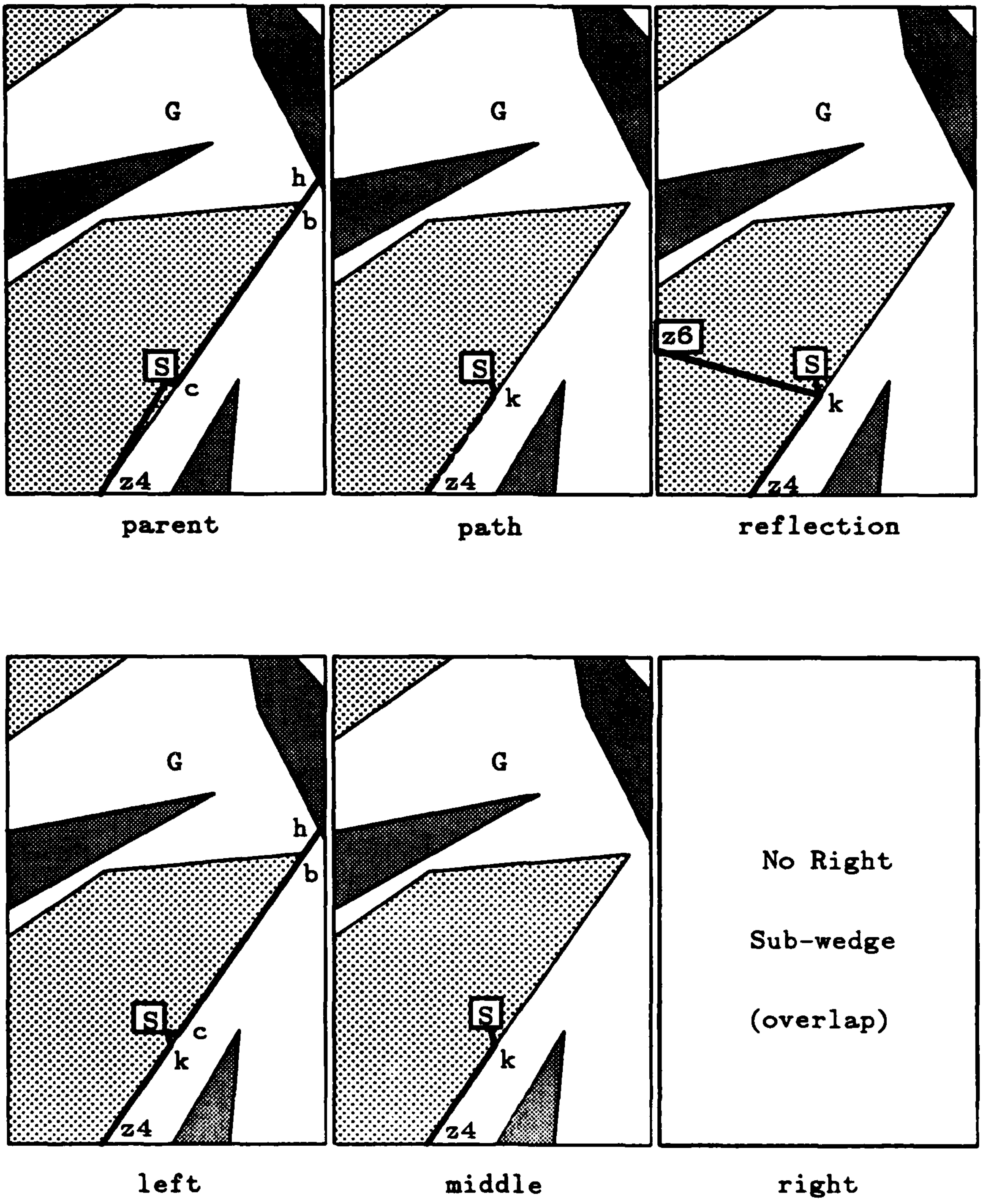
reflection

left

middle

right

Figure 164. Refinement of Wedge W7



parent

path

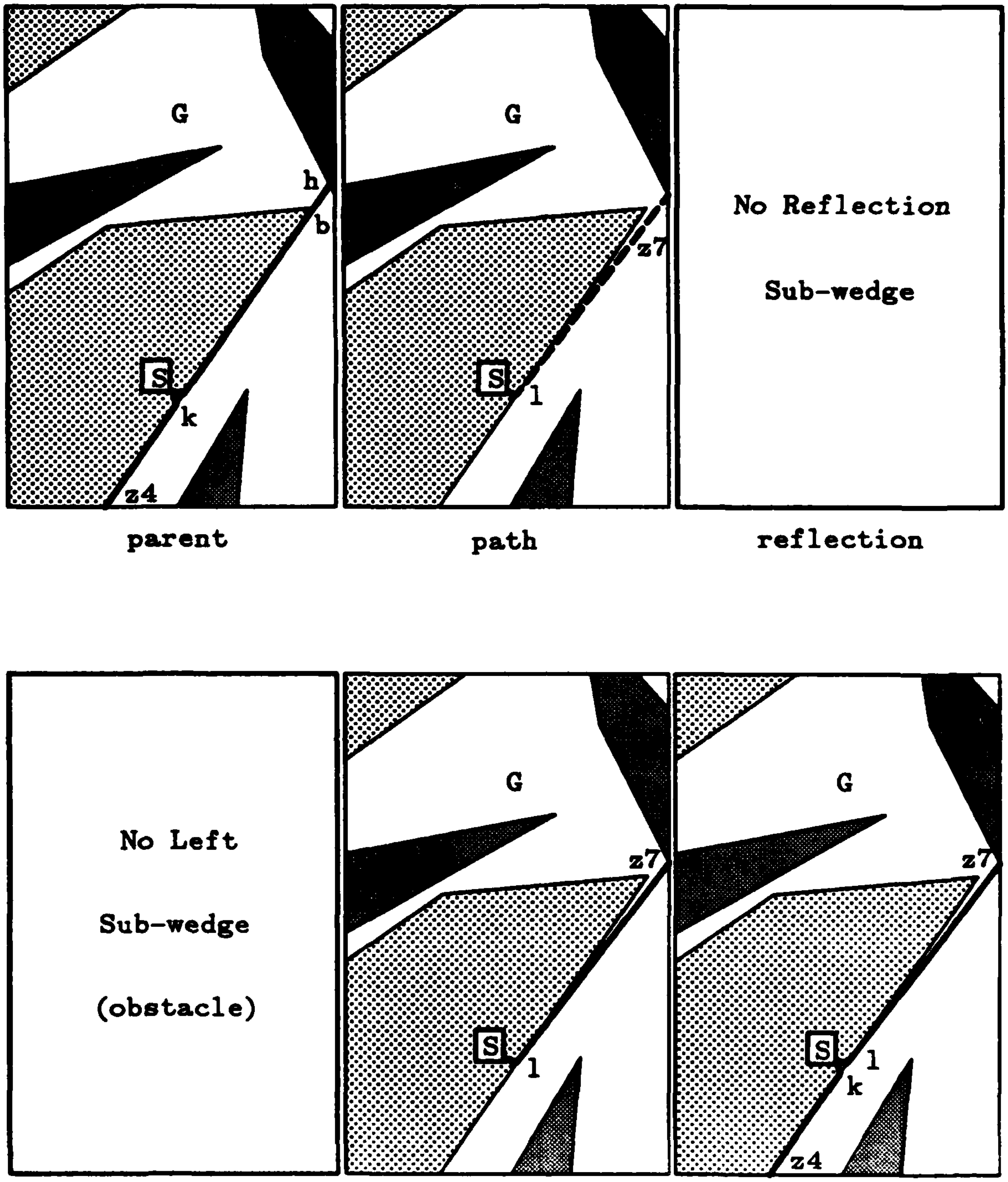
reflection

left

middle

right

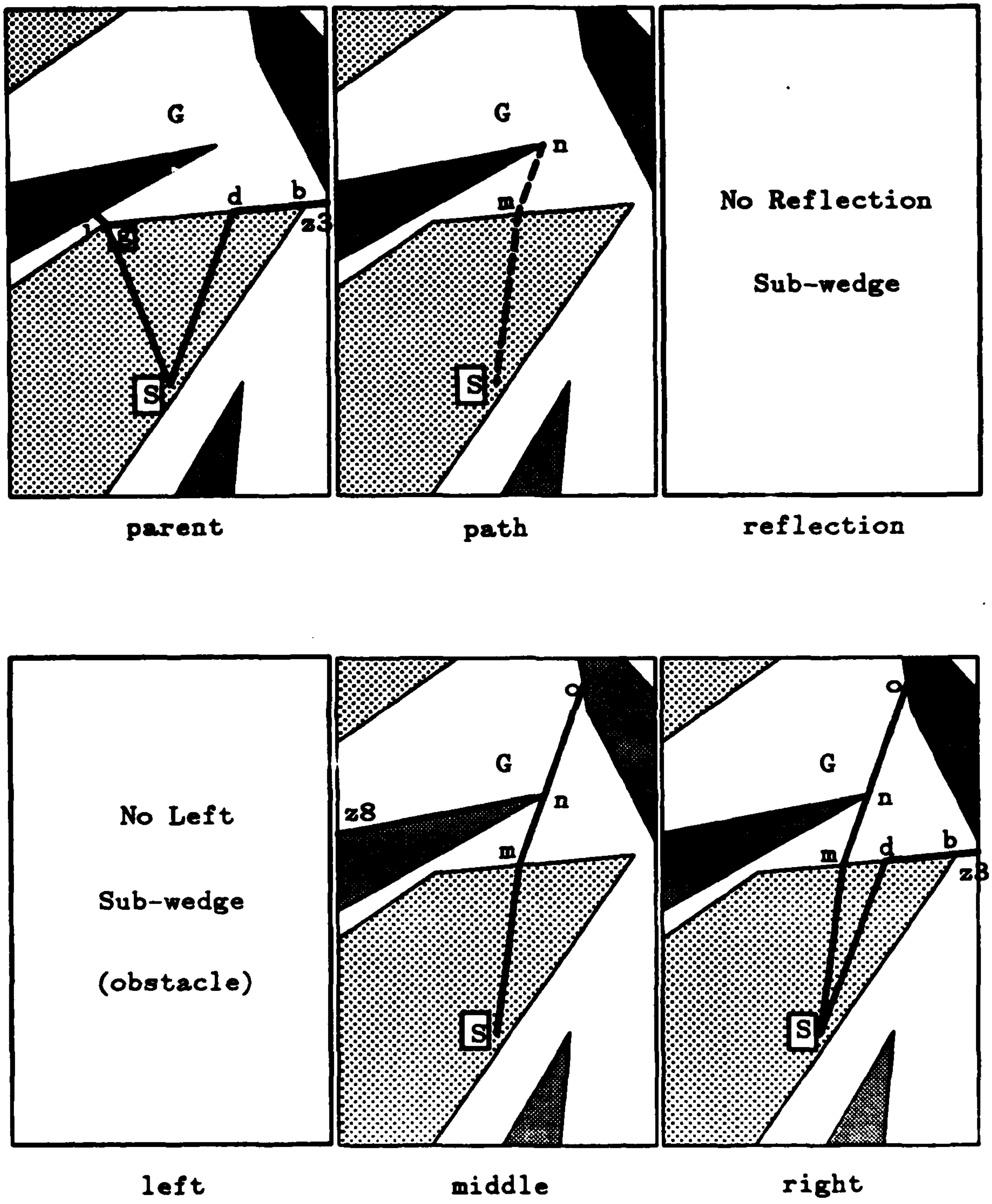
Figure 165. Refinement of Wedge W5



parent path reflection

left middle right

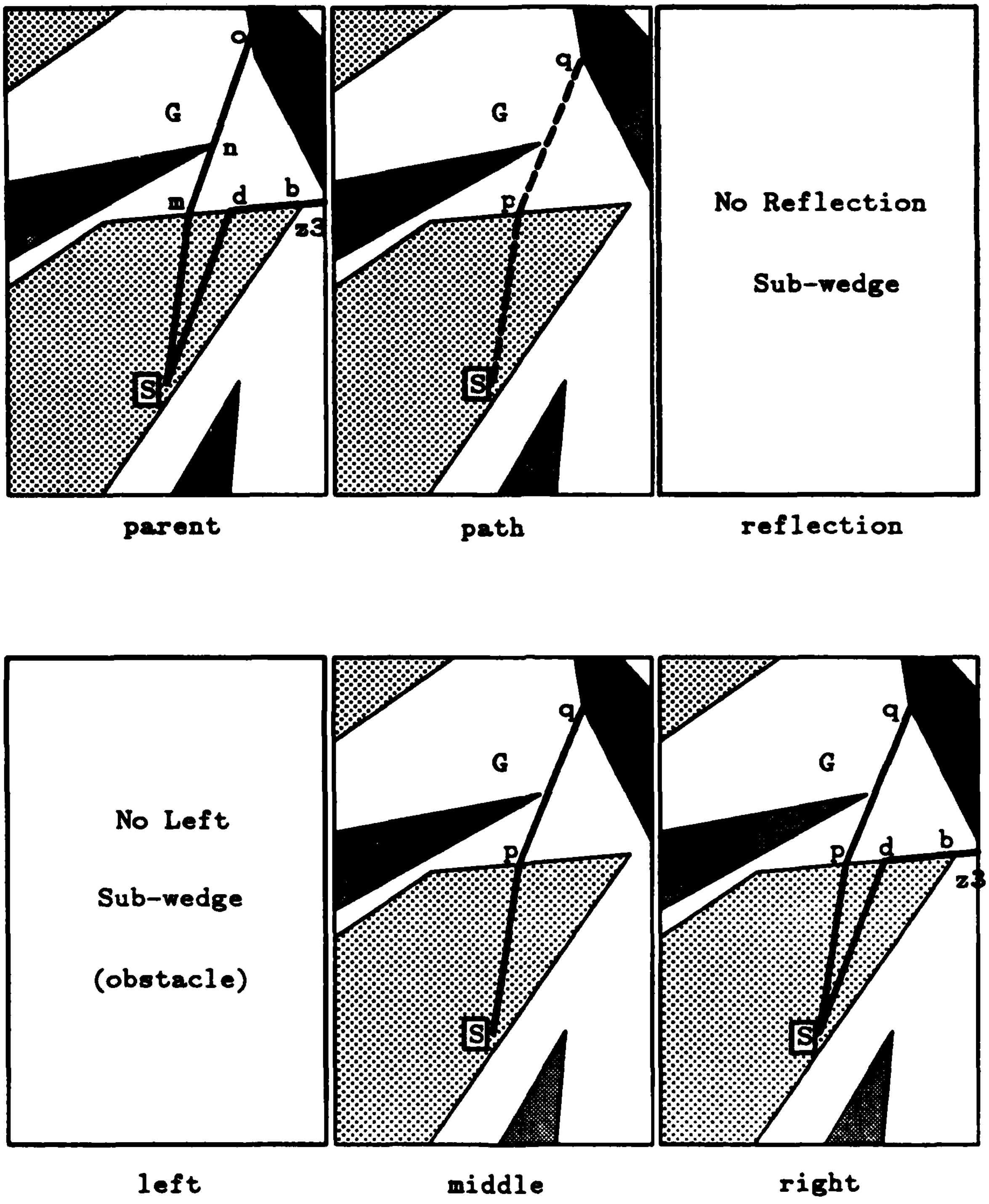
Figure 166. Refinement of Wedge W11



parent path reflection

left middle right

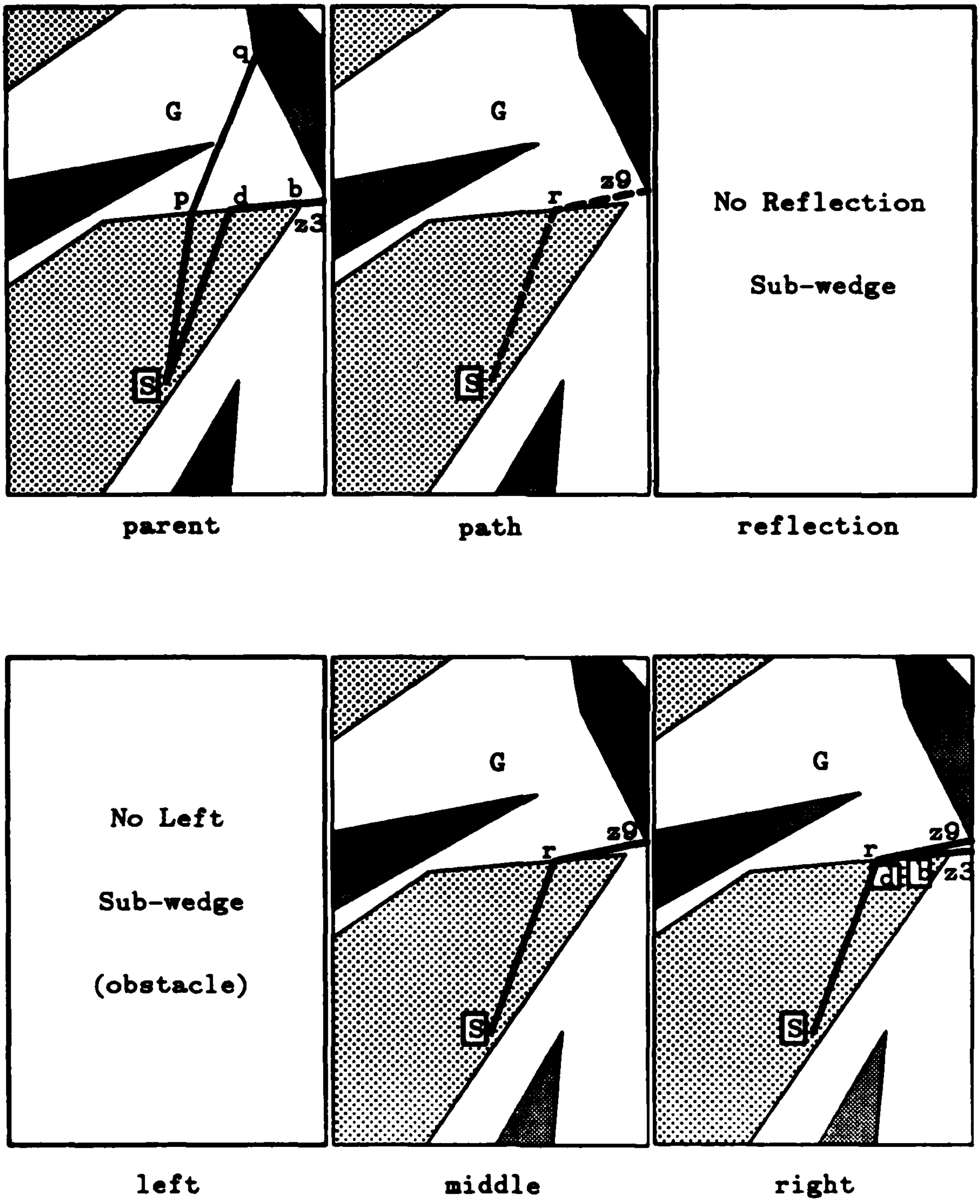
Figure 167. Refinement of Wedge W8



parent path reflection

left middle right

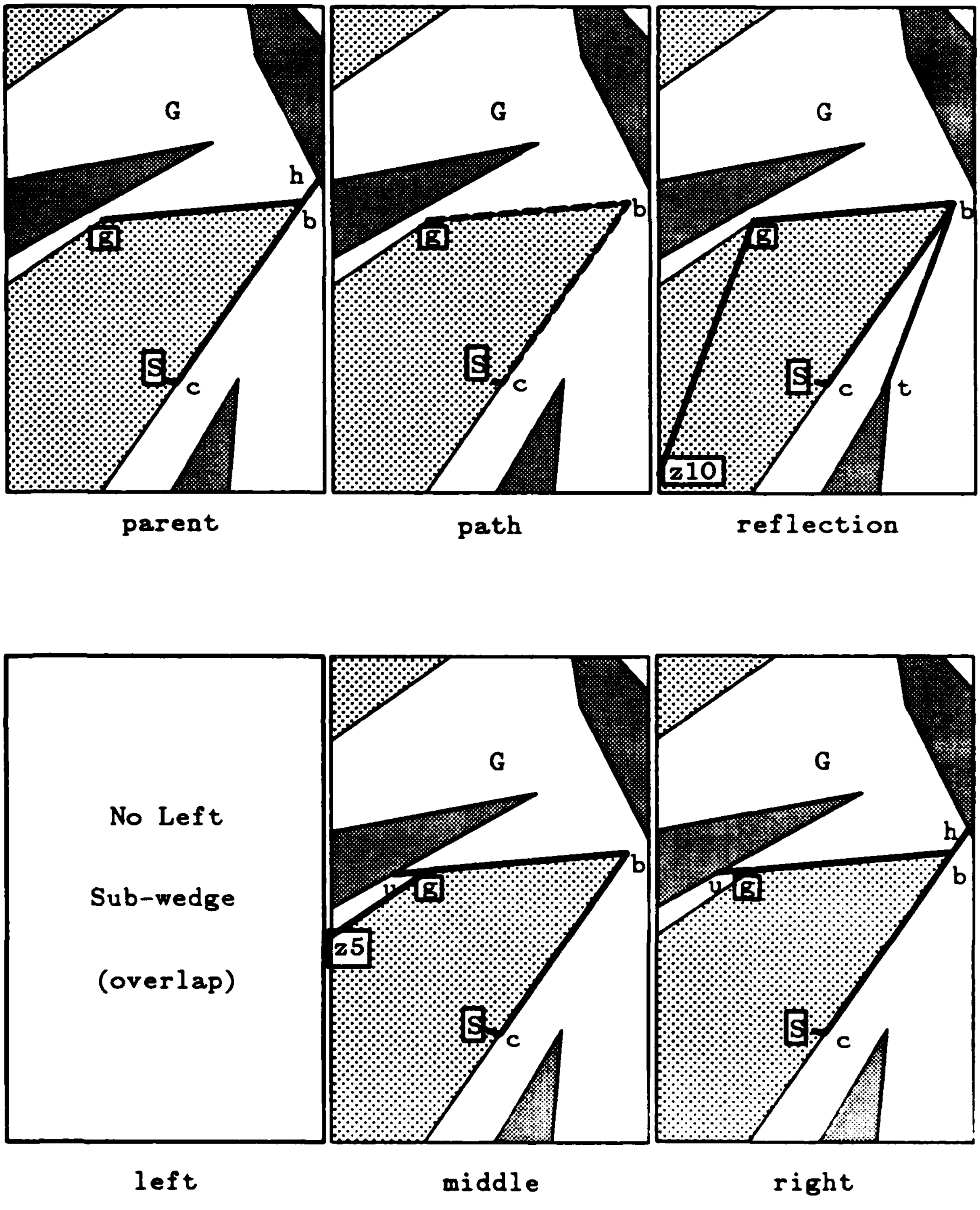
Figure 168. Refinement of Wedge W13



parent path reflection

left middle right

Figure 169. Refinement of Wedge W15



parent

path

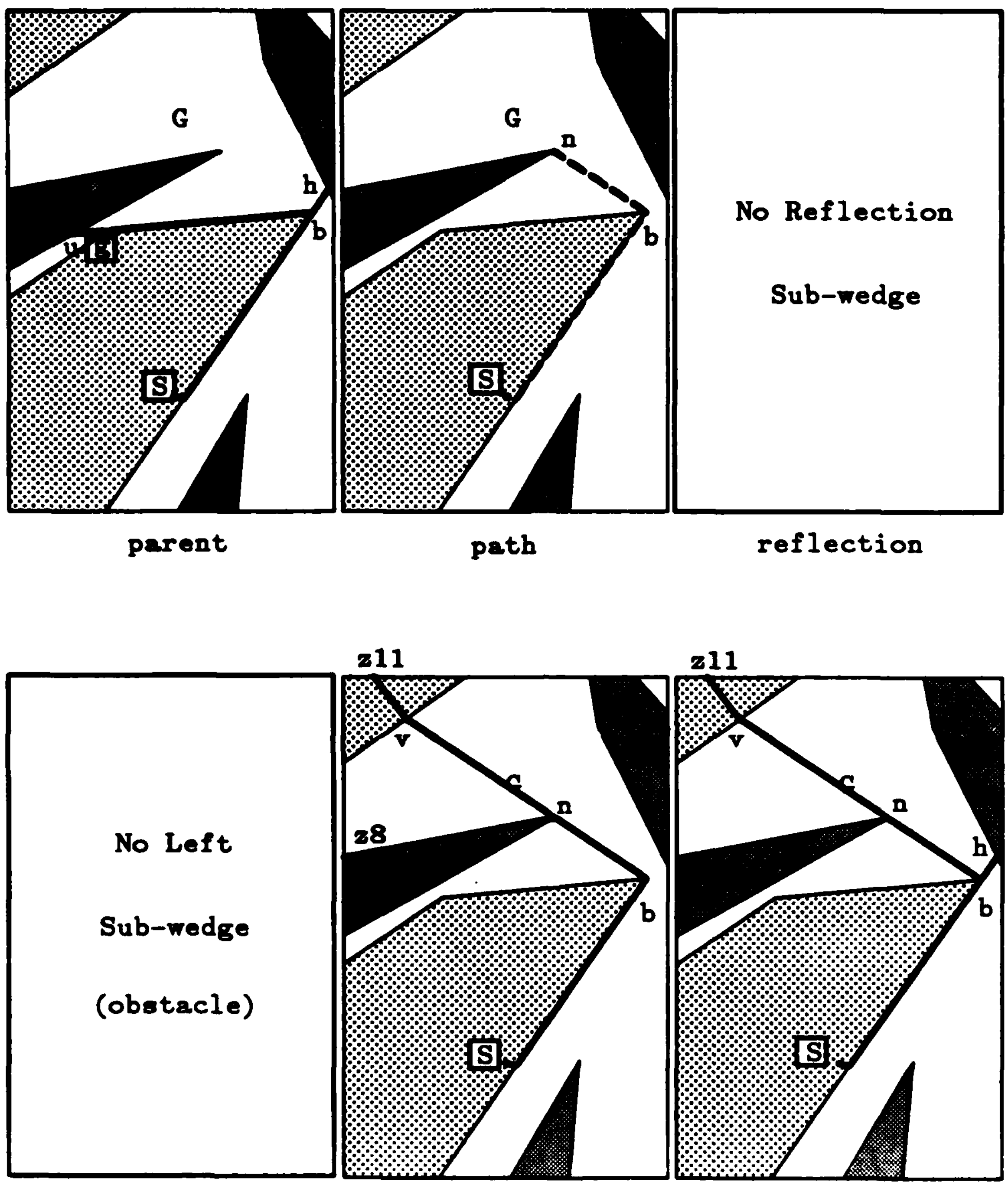
reflection

left

middle

right

Figure 170. Refinement of Wedge W4



parent

path

reflection

left

middle

right

Figure 171. Refinement of Wedge W18

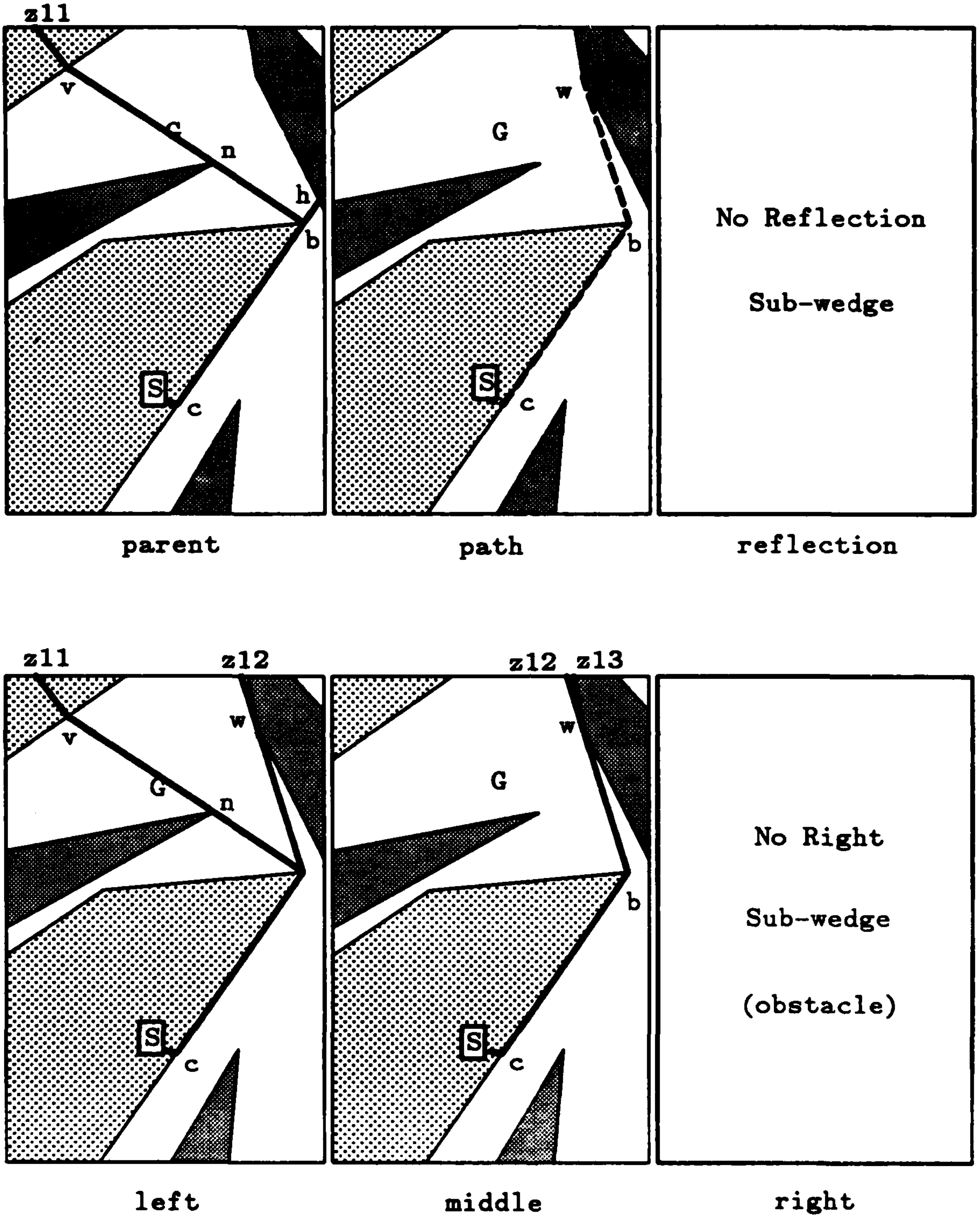
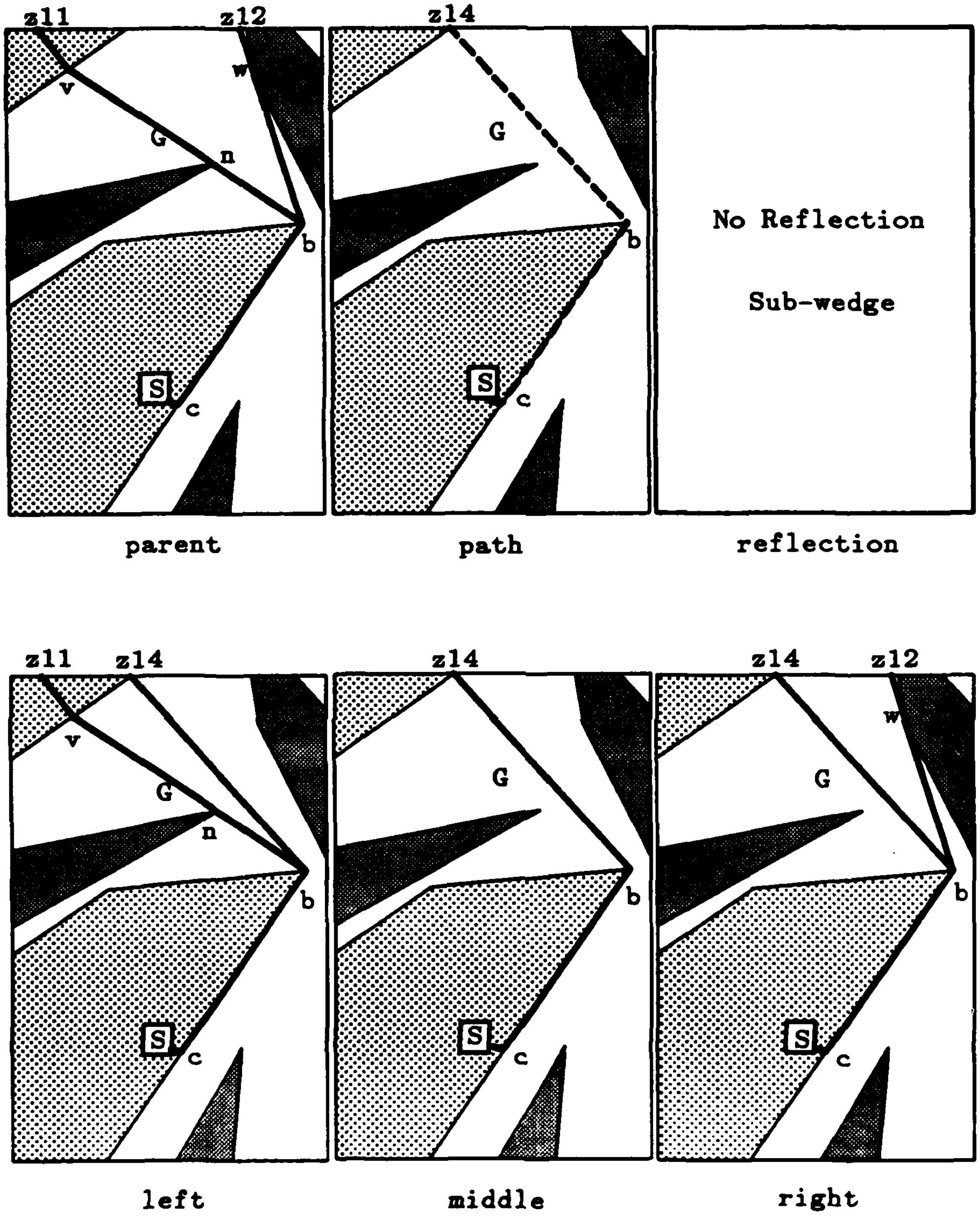


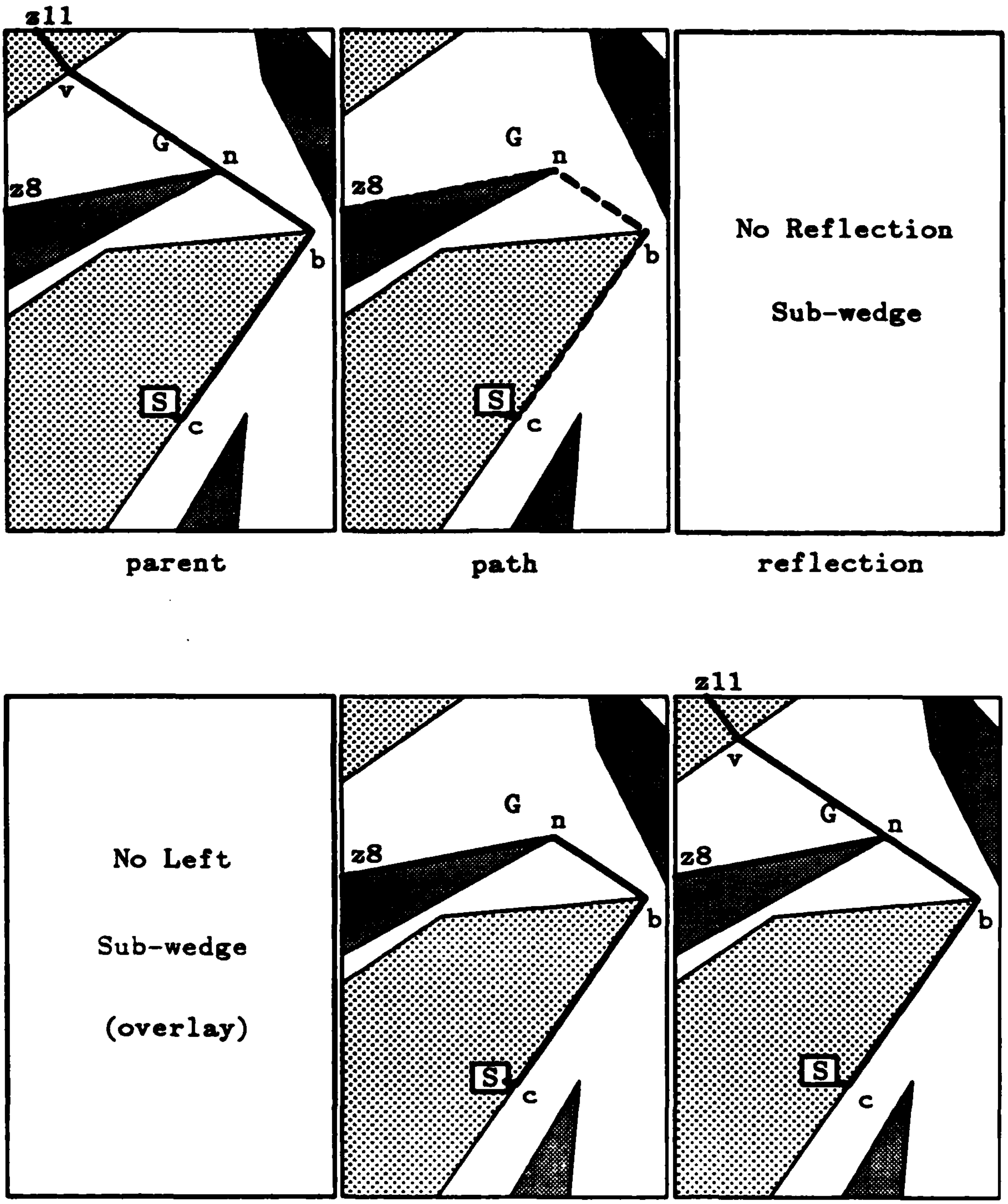
Figure 172. Refinement of Wedge W20



parent path reflection

left middle right

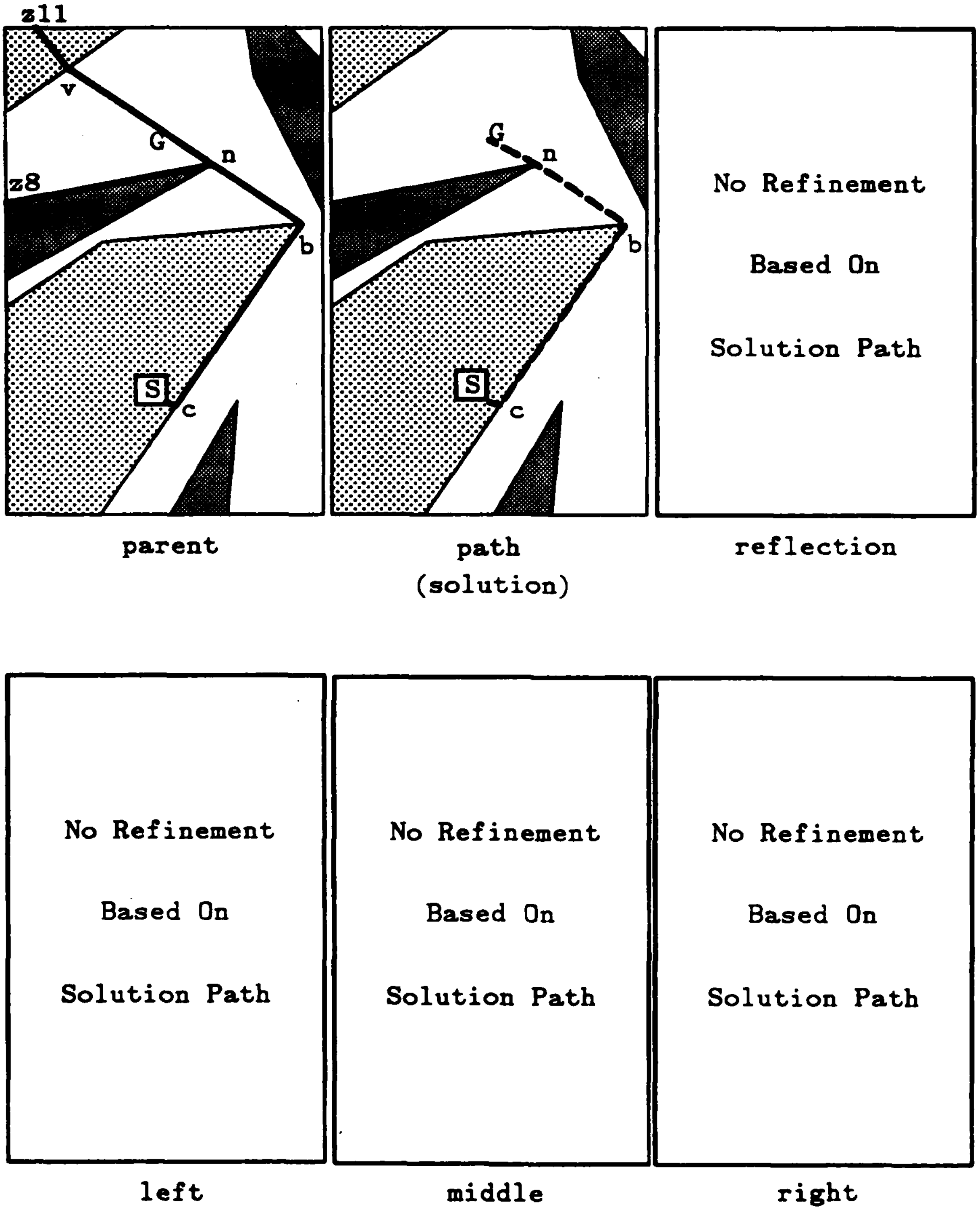
Figure 173. Refinement of Wedge W21



parent path reflection

left middle right

Figure 174. Refinement of Wedge W19



parent

path (solution)

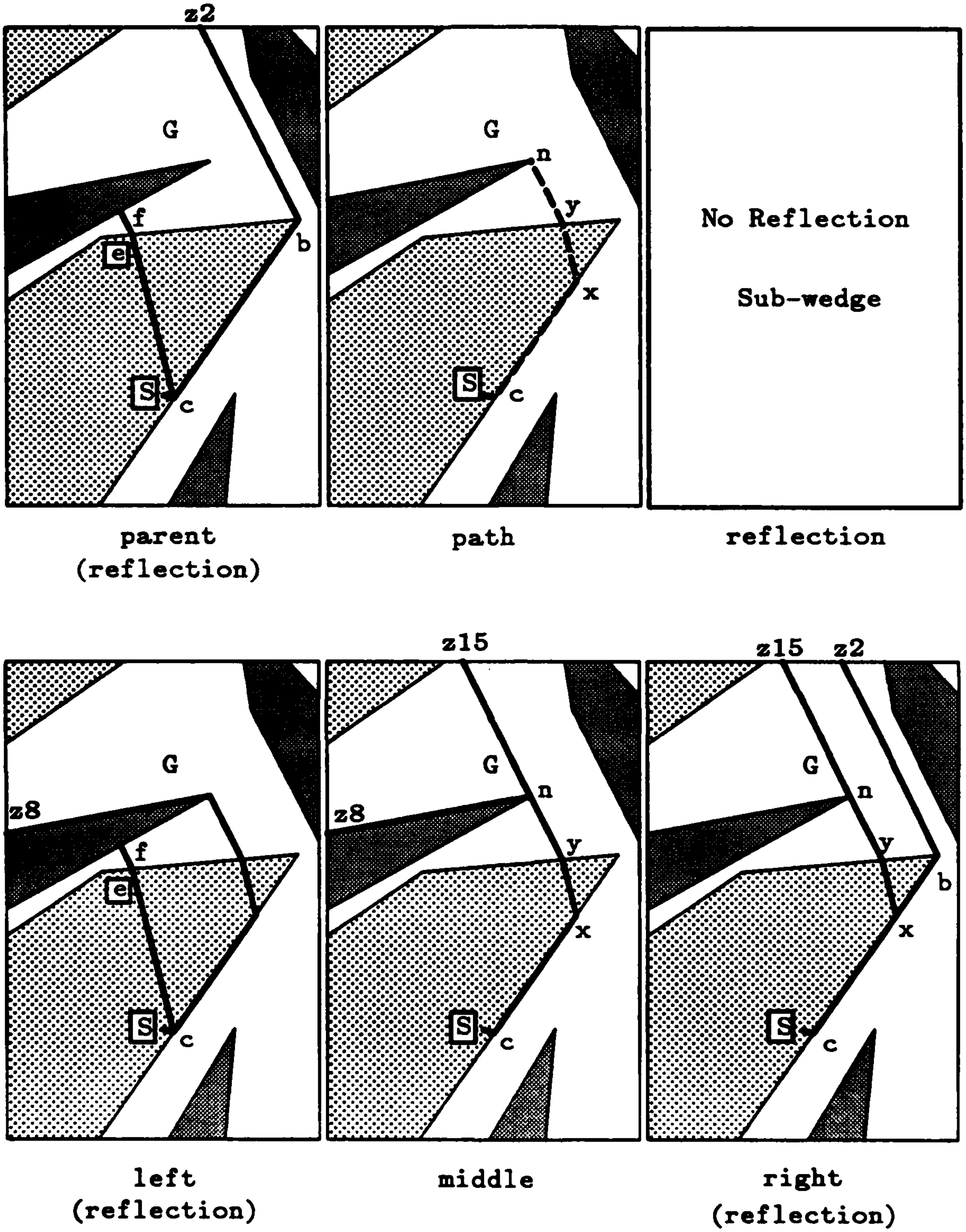
reflection

left

middle

right

Figure 175. Solution Path Through Wedge W24



parent (reflection)

path

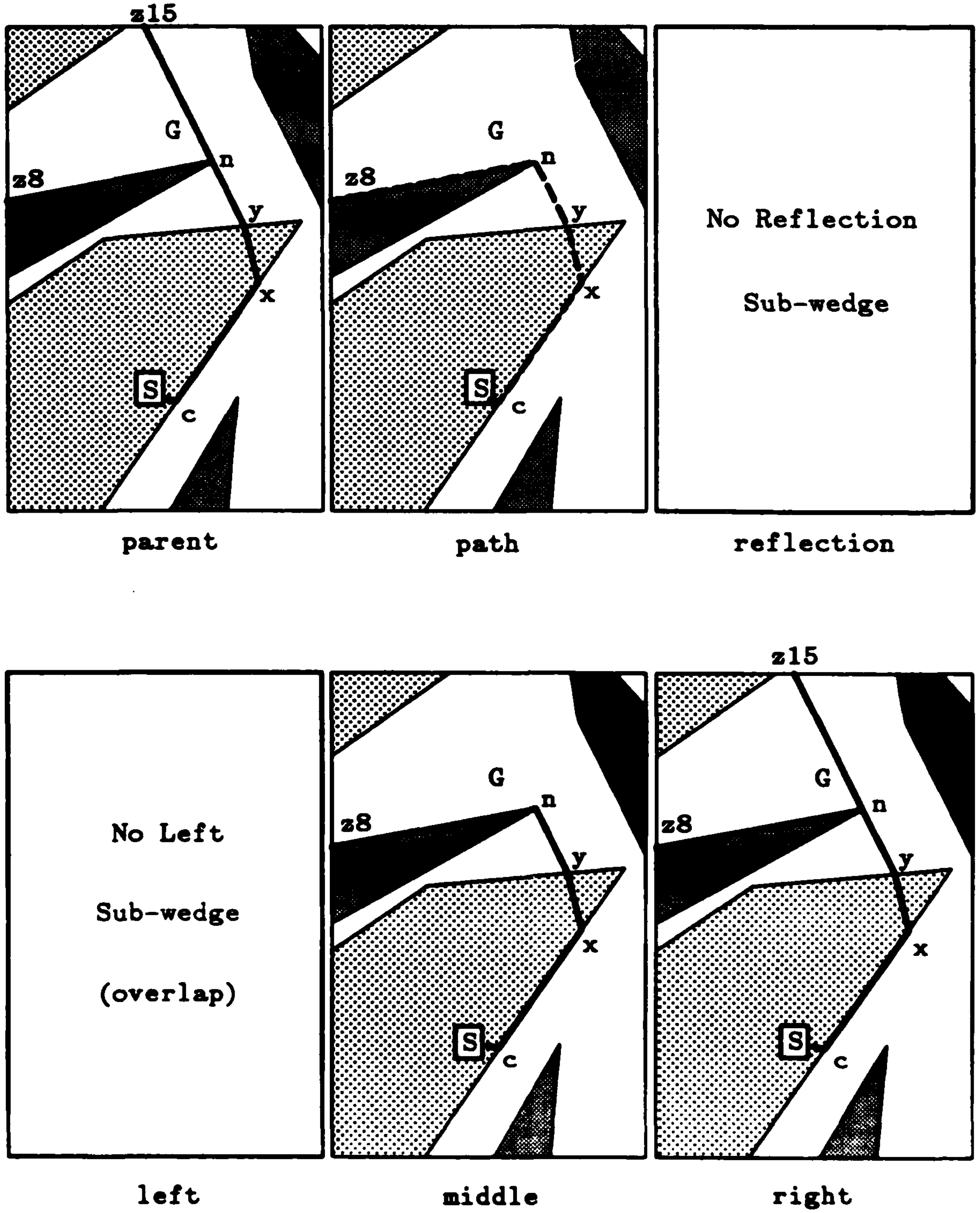
reflection

left (reflection)

middle

right (reflection)

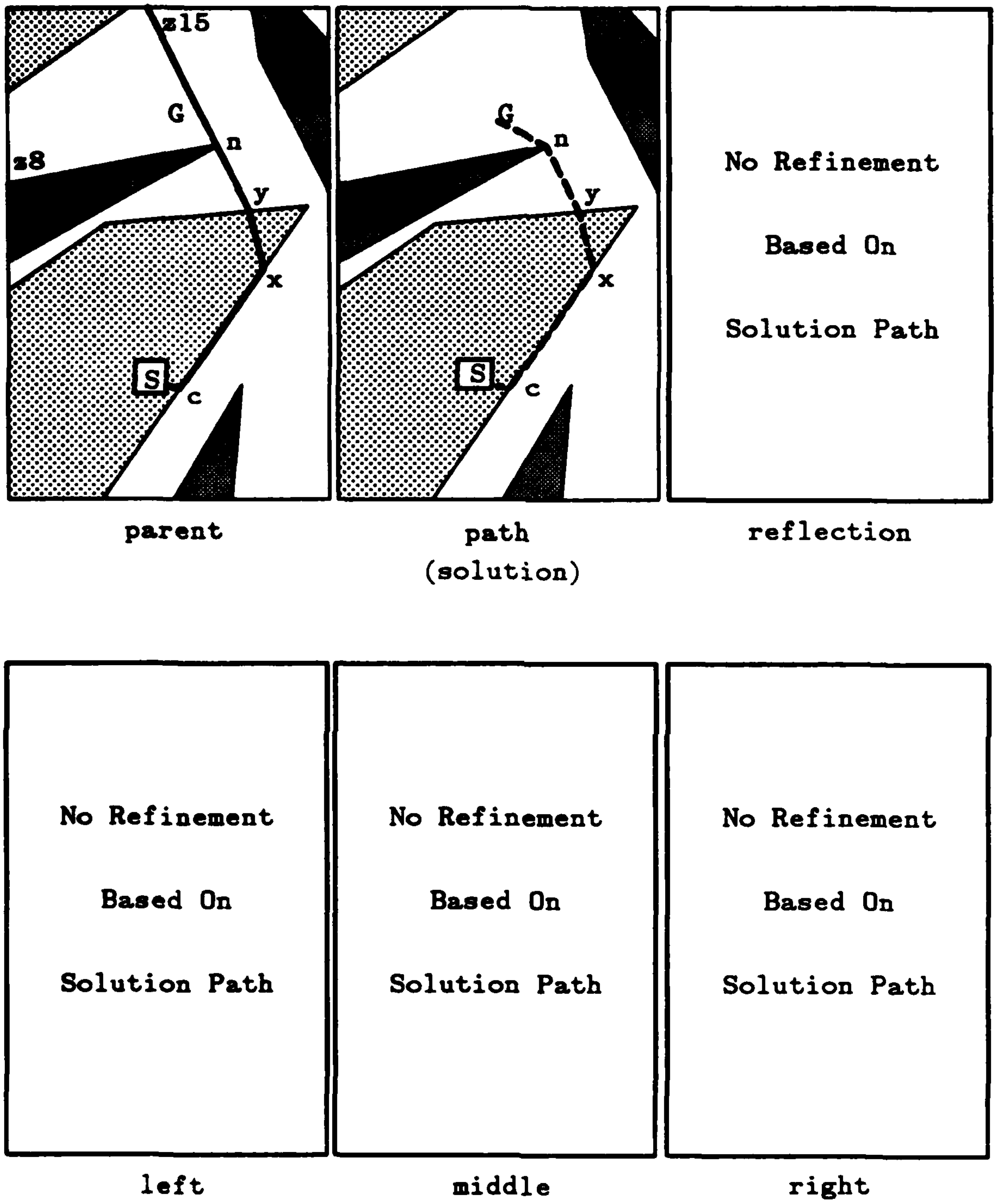
Figure 176. Refinement of Wedge RW1



parent path reflection

left middle right

Figure 177. Refinement of Wedge W25



parent

path (solution)

reflection

left

middle

right

Figure 178. A Solution Path Through Wedge W26

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Dr. Neil C. Rowe, Code 52Rp Department of Computer Science Naval Postgraduate School Monterey, California 93943	2
4.	Dr. Michael J. Zyda, Code 52Zk Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
5.	Dr. Robert B. McGhee, Code 52Mz Department of Computer Science Naval Postgraduate School Monterey. California 93943	1
6.	Dr. Uno Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
7.	Major Richard Adams, Code 52Ad Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
8.	Dr. Vincent Y. Lum, Code 52Lu Chairman. Department of Computer Science Naval Postgraduate School Monterey, California 93943	2

No. Copies

9. Dr. G.T. Howard Code 012 2
Director of Research Administration
Naval Postgraduate School
Monterey, California, 93943

10. Dr. David K. Hsiao, Code 52Hq 1
Department of Computer Science
Naval Postgraduate School
Monterey, California, 93943

11. Dr. R. K. Wood, Code 55Wd 1
Department of Operations Research
Naval Postgraduate School
Monterey, California 93943

12. Dr. Douglas Smith 1
Kestrel Institute
1801 Page Mill Road
Palo Alto, California 94304

13. Mr. Russell Davis 1
HQ, USACDEC
Attention: ATEC-IM
Fort Ord, California 93941

14. Major Robert Richbourg 3
USMA
Office of Artificial Intelligence Analysis and Evaluation
Attention: MADN-B
West Point, New York 10996

15. Dr. Joseph Mitchell 1
School of Operations Research and Industrial Engineering
Upson Hall
Cornell University
Ithaca, New York 14853

16. Major Dana Madison, Code 52 1
Department of Computer Science
Naval Postgraduate School
Monterey, California 93943

END

9-87

DTIC