Order Number 9308350

A study on the automatic path planning problem for various workspaces

Jun, Sungtaeg, Ph.D.

The University of Michigan, 1992

U·M·I
300 N. Zeeb Rd.
Ann Arbor, MI 48106

A STUDY ON THE AUTOMATIC PATH PLANNING PROBLEM FOR VARIOUS WORKSPACES

by

Sungtaeg Jun

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Information and Control Engineering)
in The University of Michigan
1992

Doctoral Committee:

Professor Kang G. Shin, Chairperson
Assistant Professor Edmund H. Durfee
Assistant Research Scientist Dr. Jun Ni
Associate Professor Michael W. Walker

To My Wife Dongmin

ACKNOWLEDGEMENTS

In preparing this thesis, I have received advice, encouragement, and support from many individuals. I have my highest regards for each one of them and thank them all. Especially, I would like to acknowledge those who have contributed directly to the successful completion of this work.

I would like to thank my advisor Professor Kang G. Shin for his consistent guidance, encouragement, faith in my ability, and financial support. His guidance kept me on the right track, his encouragement alleviated my frustrations, his faith in my ability provided constant strength, and his financial support has been critical for undertaking this research. Without him, I certainly would not have finished this work. I am indebted to Professor Michael Walker for his invaluable support for this thesis work. I also wish to thank the other members of my doctoral committee, Dr. Jun Ni and Professor Edmund Durfee, for their support in reading and commenting on this dissertation.

I would like to acknowledge my fellow students in the Ph. D. program in Electrical Engineering and Computer Science at the University of Michigan, Ann Arbor, for their assistance. I am extremely thankful to S. Bartlett and J. H. Lee for helping me prepare this thesis. X. Cui, D. M. Kim, and J. Dionese gave me much needed encouragement while preparing this thesis. Finally, I thank my family for their moral and financial support.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

LIST OF TABLES

Table

CHAPTER I

INTRODUCTION

With advances in technology in flexible manufacturing [74, 73] such as CAD/CAM, automatic task planning has become increasingly important over the last decade. The applications of this technique are diverse. Some of the most notable applications are automatic path generation for mechanical manipulators [46, 47, 83, 43] and/or autonomous vehicles [81, 26], automatic channel routing in VLSI design [54], and computer networks [12].

Typically, automatic task planning is performed in the hierarchical manner. Let's consider a hierarchy that implements a flexible manufacturing system. Fig. 1.1 shows a part of such hierarchy. In this example, an assembly process consists of a set of tasks $T = \{t_1, t_2, \ldots, t_n\}$ where each t_i represents the movement of a mechanical manipulator from one place to another while performing certain operations with its end effector. Planning the movement is carried out by an automatic path planner. Execution of the movement plan is carried out in the trajectory planning level by adding timing information. As in the case of most hierarchical planning, the automatic path planner receives necessary information from the task planner while providing information necessary to the trajectory planner. In this particular example, the path planner is the middle level of three levels in the hierarchy. As it turns out, path

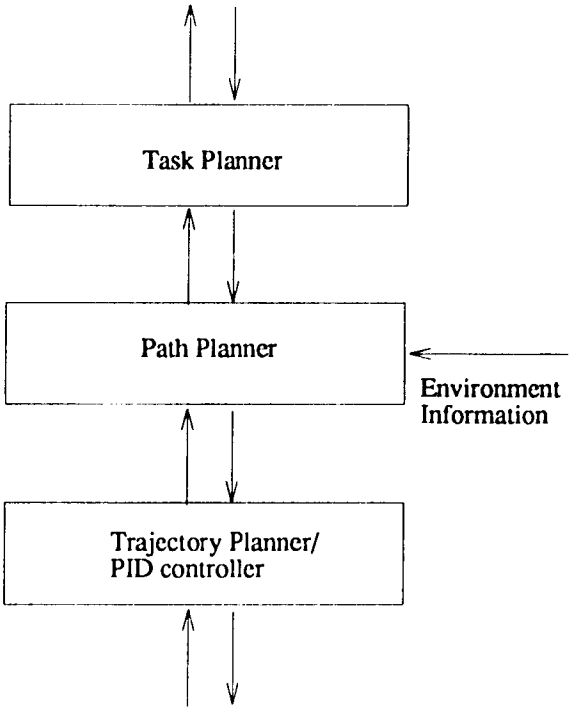


Figure 1.1: A typical hierarchy based automatic task planner.

planning is usually a low-level task, yet it requires a lot of work [71]. There have been significant progresses at the level below the path-planning level, the trajectory planner, for example by minimizing time [32] or energy consumption [33].

Being in a hierarchy, the importance and the correctness of the role of an automatic path planner is significant, because the other levels of flexible manufacturing rely on the information provided by the automatic path planning-level. The insufficient advancement at such an important level is one of the major roadblocks in achieving the goal of flexible manufacturing.

As an alternative, many of real world applications have adopted manual path planning. That is, a human guides manipulators from one location to another while recording their position and orientation. Though the use of manual path planning eliminates the difficulty of automatic path planning, it creates two new problems: 1) *consistency* and 2) *flexibility*. First, there is a consistency problem due to the

variance of humans' approaches and skills. Humans are very good at heuristics but differ greatly depending on their background. Furthermore, a human may select the use of different heuristics at different times depending on his/her emotional status. This does not necessarily mean that all heuristics are bad. However, unlike the case of the automatic path planning done by a computer, it is not easy to reproduce the knowledge of a good human. Secondly, there is a flexibility problem with the use of manual path planning. Unlike computers, humans are not good at providing many alternate paths beyond a certain limit. This can limit the flexibility of the next level of the hierarchy, i.e., the automatic task planning, due to the lack of alternatives provided by the path planning level.

Though not related to flexible manufacturing directly, there are other areas whose automation is limited by the lack of an efficient automatic path planner. For example, the use of robots may be necessary due to hazards (e.g., nuclear dump site) and the difficulty of human access (e.g., space [78] and/or deep sea exploration [16]).

For the reasons stated above, we need efficient automatic path planners. This is also an area where the limitations of conventional use of the computers are clearly shown. Traditionally, computers are designed to follow the programs provided by a human. Though almost all humans can move their able bodies, they have a difficult time explaining their reasoning. This is due to the fact that a human does not reason about moving his body. Even creatures such as bugs that are seemingly unintelligent can move their able bodies without colliding with any obstacles. This may have been achieved by nature or through learning or the combination of both.

Initial research efforts have focussed on theoretical approaches. This may have been caused by the lack of learning mechanism such as neural nets [34, 27]. Schwartz and Sharir gave an excellent survey on theoretical approaches [71]. McDermott gave a

quite cynical, but nonetheless, useful survey in [52] concentrating mostly on heuristic approaches.

One of the biggest roadblocks in achieving the general solution of this problem is its intractability. Reif has proven that certain 3D motion planning problems are PSPACE-hard [65]. Later, Canny proved that any shortest path planning in 3D is NP-complete. Since many robot movements other than mobile robots are 3D, the problem remains. Even for the case of a mobile robot, while moving from one location to another is 2D, the operation it performs at its destination may or may not be 3D or higher dimension.

Another problem associated with automatic path planning is the means of obtaining the environment information. In some applications, the environment information is explicitly given while others require the gathering of such information through sensors such as a camera [2, 66]. In many early solutions, exact environment knowledge is assumed due to the lack of advancement in sensor technology. This is valid when environment is set up so that exact information can be given to the robot in many manufacturing situations. However, even under these situations, the environment can change due to the existence of other moving robots and/or unexpected intruders. Hopcroft *et al.* [21] studied the theoretical basis of multiple moving robot environments. Lumelsky studied many cases where the environment is unknown [51, 48] or the sensor has a limited range [50].

In this dissertation, we have addressed three different environments, one 2D and two 3D environments where exact information is assumed, and we present algorithms for each specific environment. First, the classical ladder problem has been addressed in Chapter 3. The problem has been studied by many researchers as the simplest form of the Mover's problem [22]. Its minimum bound of the worst-case computa-

tional complexity is known to be $\Omega(n^2)$. Leven and Sharir proposed the most efficient algorithm known so far, with $O(n^2 \log n)$ time complexity using a free-space decomposition with trapezoids [40]. In Chapter 3, we propose an $O(n^2 \log n)$ algorithm for this problem using *Reachability Graph* based on the connectivity within a set of selected ladder positions that are critical to the path planning of the ladder. In addition, we have shown that the actual minimum bound of the worst-case computation of this problem is $\Omega(n^2 \log n)$ instead of $\Omega(n^2)$. This proves that our algorithm as well as Leven and Sharir's algorithm is computationally optimal.

Chapter 4 addresses the problem of a point path planning in a digitized 3 dimensional workspace. Although the shortest path planning of a point in a k-dimensional space is proven to be NP-Complete for $k \geq 3$, it is possible to find the shortest path in a restricted environment. Recti-linear visibility is defined in a digitized workspace. Under a digitized workspace, we have defined an equivalence relation, denoted by $\sim$, based on recti-linear visibility. Being an equivalence relation, $\sim$ partitions the workspace into a set of equivalent classes, called *regions*. It is shown that these regions are related by *dominance* relations. Upon forming a graph on the basis of this relation, it is shown that the recti-linear shortest path can be found fast thanks to regionalization instead of searching individual cells.

In some cases, this regionalization produces too many regions due to the fragmentation of the workspace. This happens when the workspace contains many small obstacles and or obstacles with many slanted edges. In Chapter 5, the path planning problem for such an environment is addressed. It is shown that our algorithm provides fast solutions using a heuristic measure called as the *probability field*. The probability field is defined using the probability that each cell of a digitized workspace becomes a destination. A formula for memoryless probability has been presented and

examples based on this are analyzed. Finally, this dissertation concludes with Chapter 6.

CHAPTER II

LITERATURE SURVEY

As mentioned in the previous chapter, automatic path planning is the problem of finding a path from one location to another. Though we have limited the problem to geometric path planning, we have to mention some of the basic solutions used in other aspects of the problem. First, we have to transform a geometric object into symbolic constraints so that computers can manipulate it. In its most symbolic form, path planning is performed in a graph, i.e., graph search.

For graph search problems, Moore [55] has suggested an algorithm to find the shortest path between two vertices in the graph. It is well known as the *Breadth First Search* (BFS) technique. The limitation of the BFS is that it can solve the shortest path planning problem for only graphs with unique length edges. Later, Dijstra [15] has suggested a $O(n^2)$ algorithm for a graph with non-unique length edges. Katoh *et al.* have proposed an algorithm to find the k shortest paths in [28] accommodating repetitive queries. The lower bound for computing the shortest path for every pair of the nodes in a graph has been proven to be $O(n^2 \log n)$ in [84].

It is difficult to apply these graph search algorithms directly to our problem due to the fact that they do not consider the presence of the obstacles or the volume of the moving object. However, many efforts have been made to transform the geometric

problem into a graph search problem by eliminating the effect of the volume of the moving object and the obstacles. Udupa [83] has used the configuration space approach in mechanical manipulator motion planning where the size of a moving object can be reduced to a point representing its configuration variables instead of cartesian positions of the mechanical manipulator. Later, Lozano-Perez refined the notation and a more general algorithm to obtain the configuration space for a polygonal object moving amidst polygonal obstacles [44]. Khatib has proposed an artificial field approach [31] where the destination generates attracting forces while an obstacle generates repelling forces. This approach is extremely useful for many applications in the area of mobile robot navigation because we can generate an artificial field as sensors detect obstacles. This approach is inherently heuristic and has become the basic idea for many heuristic applications. By contrast, Tarski's Theorem of *real closed field* [80] has been the base of many theoretical works [9, 68, 36]. Using these basic tools, the earliest attempts in solving the automatic path planning problem were made to find the shortest path in a given environment.

2.1 Theoretical Solutions

Lee and Preparata [38] have proposed an $O(n \log n)$ time algorithm to find the shortest path between two points inside a polygon. They have used the *Delaunay triangulation* technique [13]. Later, Guibas *et al.* have improved the algorithm to handle repetitive queries in [17, 18].

The L_1 shortest path, sometimes known as the *Manhattan* path, has been studied by DeRezende, Lee, and Wu [14] in the presence of orthogonal barriers. Their solution is based on the BFS. The *Euclidean* shortest path, L_2 shortest path, has been proposed in [38] using the *Shortest Path Map* (SPM). In the SPM, the obstacles are

limited to vertical lines. Consider a point A and a vertical line $\overline{BC}$ in the workspace. Then, there are two different regions in the workspace: (1) the region that is visible from A and (2) the region that is not. For any point D in (2), the shortest path between A and D is either $\overline{ABD}$ or $\overline{ACD}$. That is, (2) is further divided into two regions. Any vertical line inside a region will further divide the workspace. Thus, the entire workspace is divided into several regions. The limitation of the SPM is that it only accepts vertical lines as obstacles. For a more general situation, the visibility graph (VG) has been used in [10, 45, 11, 75]. Unlike the SPM where the obstacles are limited to orthogonal barriers, the VG assumes the obstacles to be polygons. The VG is based on the observation that when two points are not visible from each other, the shortest path between them contains at least one vertex of an obstacle. Hence, the distances between all the visible pairs of vertices of the obstacles are calculated *a priori* and transformed into a graph. The actual search for the path is carried out using the Dijstra's graph search algorithm on this graph.

Clackson *et al.* [11] have extended the definition of visibility graph to L_1-metric system and proposed an $O(n \log^2 n)$ time algorithm to construct the L_1 visibility graph. They also applied the same algorithm to construct a 3D L_1 visibility graph. However, they failed to show the actual application of the 3D visibility graph in the shortest path planning problem.

Shortest path planning for a moving disc was developed by Chew in [10]. The algorithm first converts the obstacles into a configuration space and constructs the visibility graph based on the configuration space. The shortest path was searched using a graph search algorithm on this graph. The overall complexity of the algorithm is $O(n^2 \log n)$.

Later, Canny [9] proved that any shortest path planning in 3D is NP-Complete

and proposed the *roadmap* algorithm which is a one dimensional subset of the configuration space of a robot manipulator using the generalized (multivariable) resultant for a system of polynomials and achieved single exponential time complexity. The roadmap algorithm has achieved an exponential speedup over an existing algorithm [36] which has a double exponential growth in time with the number of variables. Due to this excessive computational requirement, we consider solutions to a simpler problem, i.e., Find Path Problem (FPP), known as the *Generalized Mover's Problem* [22].

Most FPP algorithms depend on the model of a moving object and/or its environment. The simplest form of a moving object is a point. As stated before, the goal of reducing a moving object with some physical volume to a point is achieved by the configuration space approach. After reducing the moving object we may apply some of known algorithms such as the visibility graph or regionalization to obtain a path.

The next simplest form of modeling a moving object is a disc. O'Dunlaing and Yap [59] developed an algorithm that detects the existence of the path while moving a disc amidst polygonal barriers. Their algorithm is to transform the workspace into a Voronoi diagram where each vertex of the diagram is represented by its coordinates and each edge of the diagram by a parabolic function. For each edge of the diagram, its minimum clearance from the nearest obstacle edge is precalculated to determine whether the disc can be slid through. Path planning is performed using a depth-first search on this Voronoi diagram. Overall complexity of the algorithm is $O(n \log n)$.

The main advantage of using the Voronoi diagram path planning is the safety of the generated paths. The path generated using the Voronoi diagram always move the disc through the middle of two obstacle edges. In contrast, the paths generated using the visibility graph usually touches a surface or vertex of an obstacle. Though

it can find somewhat a safer path, the resulting path can be very long. Suh and Shin [79] have proposed an algorithm that finds the minimum weighted distance-clearance using a Voronoi diagram and dynamic programming.

Moving a line segment has been extensively studied by Schwartz and Sharir [68, 69]. Their algorithm converts the workspace to a map of critical curves. The critical curves are defined as an area in which the line segment can move with only limited motion. They classified the critical curves into three different models according to the motions allowed in the area. Upon identifying all the critical curves the algorithm works as follows:

1. Find the intersection points of all the critical curves with each other and with the walls.

2. Sort the intersection points along each critical curve.

3. Find the crossing map according to crossing rules.

Schwartz and Sharir [69] implemented the same algorithm in 3D with $O(n^8 \log n)$ time complexity. Later, Ke and O'Rourke [29] improved the algorithm so as to make it have $O(n^6 \log n)$ upper-bound complexity and $O(n^4)$ lower-bound computational complexity.

Shortly after that, O'Dunlaing *et al.* proposed $O(n^2 \log n \log^* n)$ algorithm using a generalized Voronoi diagram. This was promptly superseded by Leven and Sharir who proposed an $O(n^2 \log n)$ algorithm using workspace decomposition with trapezoids [40]. Sifrony and Sharir [76] proposed another algorithm to solve the ladder problem. Although it has the same computational complexity of $O(n^2 \log n)$ as Leven and Sharir's algorithm, it improves the actual computation time under sparse obstacle positioning. The minimum bound of the worst-case computation time known so

far is $\Omega(n^2)$ as shown by O'Rourke in [61]. However, no algorithm as yet has achieved this minimum bound.

As for a more general moving object, Lozano-Perez and Wesley studied this problem and proposed an $O(n^3)$ algorithm for moving a convex polygon [45]. Initially, this algorithm was limited to translational movements of the convex polygon. Later Brooks and Lozano-Perez improved the algorithm and proposed a *hierarchical subdivision* algorithm which allows the rotation of a moving object [7]. We will discuss more about this latter work in the following section, as the algorithm is more or less heuristic.

Leven and Sharir [42] used a generalized Voronoi diagram to obtain an $O(n \log n)$ algorithm to solve the path planning problem for purely translational movements of a convex polygon amidst polygonal obstacles. Later, Leven and Sharir [41] showed the number of free critical contacts to be $O(kn\lambda_s(kn) \log kn)$, where k is the number of edges of the moving polygon, and $\lambda_s(n)$ is an almost linear function of n. By combining these two results, Kedem and Sharir [30] obtained an $O(n^2\lambda_s(n) \log n)$ algorithm to solve the path planning problem for moving convex polygons.

As for motion planning of multiple moving objects, Schwartz extended his initial work [68] on the ladder problem and proposed an $O(n^3)$ algorithm for two moving discs [70]. Hopcroft, Schwartz, and Sharir [21] proved the PSPACE-hardness of a general solution for multiple moving discs while Spirakis and Yap [77] showed the strong NP-hardness of the motion planning problem for eight or more moving discs.

2.2 Heuristic Solutions

While efforts on the theoretical aspect of path planning were successful with varying degrees, other researchers emphasized the heuristic aspect of the problem

[5, 6, 24, 7, 87]. This is mainly because of the lack of general solutions in 3D workspace. To remedy the difficulties involving 3D, there have been attempts to convert the free space into somewhat manageable form so as to make the FPP easier. Brooks [5, 6] proposed a generalized cone to represent the free space while the Octree representation [39] was used by others [23, 26]. Though the Octree is quite useful for representing 3D objects, it does require a general search algorithm to construct a path. In [26], Kambhampati and Davis used the A^* algorithm to construct a path for a mobile robot. Since the search was not efficient, they have proposed a pruning method according to the grayness[1] of a subtree.

After their work that limits the mobility of the moving polygon to translation [45], Brooks and Lozano-Perez proposed a hierarchical subdivision algorithm [7]. Unlike the previous work, the subdivision algorithm allows not only translation but also rotation of the moving polygon. However, the subdivision algorithm is still heuristic since the number of recursive subdivisions is limited until a certain accuracy is achieved. Later, Zhu and Latombe [87] improved the algorithm using *constraint reformulation* and a new hierarchical search with failure recording to achieve significantly faster computation.

Another useful method of workspace representation is the vector field approach proposed in [62]. It transforms the workspace information into a field of vectors pointing to the correct directions to travel. The major problem of this method is the precomputation of the vector field of the entire workspace and the vast amount of information to be stored. To remedy this problem, Miller and Slack revised the algorithm to dynamically compute the vector field [53].

Khatib [31] proposed an artificial potential field Approach where the workspace

[1] Portion of a subtree that is occupied by the obstacles.

is transformed to a mathematical field so that obstacles generate repelling forces and the destination generates attracting forces. This method is quite useful for real-time applications because a robot can simply follow the most attractive line and is often used in mobile robot motion planning [81, 82, 35]. It is shown that the artificial potential field approach has the drawback of getting into local minima [35, 24]. Borenstein and Koren [4] proposed a vector-force field algorithm by integrating two known concepts: certainty grids for obstacle representation [56], and potential fields for navigation. However, it does not solve the problem completely because a certainty grid is the representation of inaccurate sensory data about obstacles.

A gradient-field approach proposed by Payton [63, 64] deals with this problem by calculating the cost of each grid cell of a digital map. The cost of each cell is based on the score obtained by applying a search algorithm such as A^* [19], or Dijkstra's algorithm [15]. Though the gradient-field approach is useful in a known environment, it cannot deal with a changing environment efficiently due to its excessive computational requirements. Zhao [85] proposed an algorithm that can deal with both known and unknown environments using a heuristic-search method (recovery algorithm) based on the $A*$ algorithm. It was shown that the efficiency of Zhao's algorithm largely depends on the scale factor of the map.

In many cases, it is very difficult for a path planner to know its exact surroundings, often due to the existence of other moving robots [72], or the sudden change of the environment caused by errors and/or intruders. In some cases, the environment is too large to be modeled. The most attractive means of gathering environment information are vision, tactile sensors, and/or range finders. The survey of this area is omitted as the topic is beyond the scope of this dissertation.

Lumelsky has worked extensively on automatic path planning in an unknown

workspace. In [51], Lumelsky and Stepanov introduced two new algorithms, **bug1** and **bug2**, which search for the path of a mobile robot in an unknown territory. In [48], Lumelsky showed that the above work can also be applied to mechanical manipulators after proper modification. Later, he [49] compared the path lengths generated by **bug1** and **bug2** with those generated by traditional maze search algorithms [60] while showing the effect of the range of sensors in [50]. In [67], Sankaranarayanan and Vidyasagar introduced **alg1** by improving **bug2** by showing and removing the condition where **bug2** may go into an infinite loop.

CHAPTER III

ON THE LADDER PROBLEM

3.1 Introduction

This chapter addresses the problem of moving a ladder in 2D (called the *ladder problem*) amidst polygonal obstacles. The ladder problem is to find a continuous motion path of a ladder from an initial position to a final position without violating a set of geometric constraints imposed by the polygonal obstacles. We assume that both the ladder and the polygons are rigid bodies and do not allow bending or penetration. The problem has been studied by many researchers [68, 40] as one of the simplest form of the *General Mover's* Problem. In [68], the free space is partitioned into several 3D manifolds of free space called *critical curves*. By forming a connectivity graph among the critical curves, the free space is decomposed into a set of adjacent connected cells. This solution has $O(n^5)$ worst-case time complexity where n is the number of edges of the polygon obstacles. Another approach to this problem has been proposed in [59] using a *retraction* technique. It solves the problem by retracting the free space into 1-dimensional subspace called a *generalized Voronoi diagram* [57, 58]. It was shown that the algorithm can be implemented in $O(n^2 \log n \log n)$ time complexity. Later, an improved algorithm has been proposed in [40] by decomposing the free space with trapezoids. The resulting computational

complexity is reduced to $O(n^2 \log n)$.

In this chapter, we present an algorithm which decomposes the free space based on some ladder positions that are critical to path planning. Upon forming a graph using these positions, *Reachability Graph* (RG), we can decompose the entire workspace based on a finite number of critical positions. It is shown that our algorithm runs at the computational complexity of $O(n^2 \log n)$ where n is the total number of vertices of the obstacles. This is the same computational complexity achieved in [40] and is very close to the theoretically minimum of $\Omega(n^2)$ shown in [58]. This theoretical minimum is based on the $n^2\alpha(n)$ nodes obtained from the Voronoi diagram. Later in this chapter, we will prove that this minimum is not achievable with any non-heuristic motion planner. Specifically, it will be shown that the minimum bound of a non-heuristic motion planner for the ladder problem has the computational complexity of $\Omega(n^2 \log n)$. This in turn proves the optimality of both our algorithm and the one proposed in [58] for two different types of motion planner.

This chapter is organized as follows. In the following section, we introduce the environment and the terminologies used in our problem formulation and solution, and formally define the RG. Section 3.3 presents an algorithm necessary to construct the RG and the analysis of the algorithm together with an example. In Section 3.4, we will discuss the ladder problem in general and provide an analysis of the minimum bound of the worst-case computational complexity of the ladder problem. This chapter concludes with Section 3.5.

3.2 Definition of the Reachability Graph

Before defining the RG, we will briefly describe the environment and the terminologies used in this chapter. Let's consider a ladder $\overline{A_1A_2}$ in a 2D workspace. One

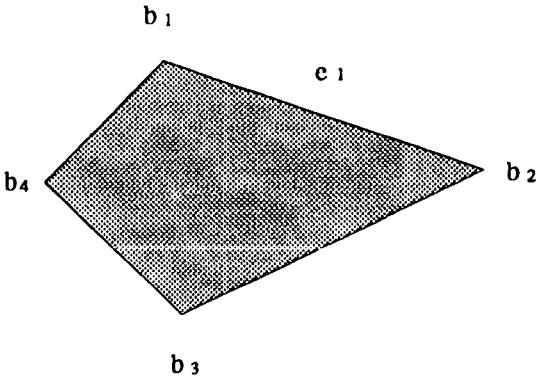


Figure 3.1: Representation of a vertex and its neighboring vertices.

end, A_1, of the ladder is marked as its reference point. The workspace is cluttered with polygonal obstacles which consist of a set of vertices $B = \{b_1, b_2, \ldots, b_n\}$ and edges connecting these vertices. Notice that these vertices also include the boundaries of the workspace. We assume all the polygons to be simple, i.e., there are only two edges meeting at each vertex. This assumption should not reduce the generality of our algorithm as any complex polygon can be represented by a set of simple polygons. We denote $cw(b_i)$ to be the clockwise vertex of b_i that shares an edge with b_i. Similarly, $ccw(b_i)$ denotes the counter-clockwise vertex of b_i. An edge e_i refers to the edge connecting b_i and $cw(b_i)$. In the example shown in Fig. 3.1, $cw(b_1)$ denotes b_2 while $ccw(b_1)$ denotes b_4. Also, e_1 denotes the line segment $\overline{b_1 cw(b_1)}$, i.e., $\overline{b_1 b_2}$. An edge e_i also defines an inequality predicate $f_{e_i}(x)$ where $f_{e_i}(x) < 0$ represents a point x that is located on the *obstacle side* of e_i. Otherwise, the point x is said to be on the *free side* of e_i. The same point is said to be *inside* of the obstacle if the point is completely surrounded by a set of edges and is on the obstacle sides of all the obstacle edges surrounding it.

A position $c = \{c_p, c_\theta\}$ of the ladder $\overline{A_1 A_2}$ represents the location (c_p) of its reference point (A_1) and the orientation[1] (c_θ) of the ladder. The *dual* of a position

[1] the angle between $\overline{A_1 A_2}$ and the horizontal line

$c = \{c_p, c_\theta\}$ is referred to as $d^*(c)$ is $\{c'_p, c'_\theta\}$ where

$$c'_p = c_p + R(c_\theta) \mid \overline{A_1 A_2} \mid$$

$$c'_\theta = -c_\theta$$

where $R(c_\theta)$ is a rotation matrix and $\mid \overline{A_1 A_2} \mid$ is the length of the ladder. Basically, the dual of a position represents the same line segment expressed using the other end, A_1, of the ladder as its reference point. We use $A_1(c)$ and $A_2(c)$ as the respective positions of A_1 and A_2 when the ladder is at position c. The ladder at a position c corresponds to a line segment $\overline{a_1 a_2}$ where $a_1 = A_1(c)$ and $a_2 = A_2(c)$. The line segment corresponding to the ladder located at c is denoted by $l(c)$.

A position c of the ladder is said to be *free* when the corresponding line segment $\overline{a_1 a_2}$ neither intersects, nor stays completely inside of, any polygon obstacle. Collectively, the set of all free spaces are referred to as FS (free space). FS was shown in [ShSw83] to be three dimensional manifolds. A path of the ladder is defined as a contiguous sequence of positions in FS that the ladder will go through. In particular, a position of the ladder is said to be *on* a vertex v when a part of the ladder is touching v. Similarly, the ladder is said to be *on an obstacle edge* when a part of the ladder is touching the edge. When two obstacle vertices u and v can be connected by a straight line without intersecting any other obstacle edges, the two vertices are said to be *visible* from each other and such a relation is denoted as uVv or vVu.

Let's consider a simple movement of the ladder (see Fig. 3.2). The visibility between the origin and destination positions, a_1 and a'_1, of A_1 is blocked by obstacle edges e_1, e_3, e_4, and e_5. Among these edges, e_1 is the closest to the position (a_1) of A_1. Ignoring all other obstacle edges, the only possible way for the ladder to go around the edge $e_{1,2}$ is to pass through the extension of either $\overline{a'_1 v_1}$ or $\overline{a'_1 v_2}$. We refer

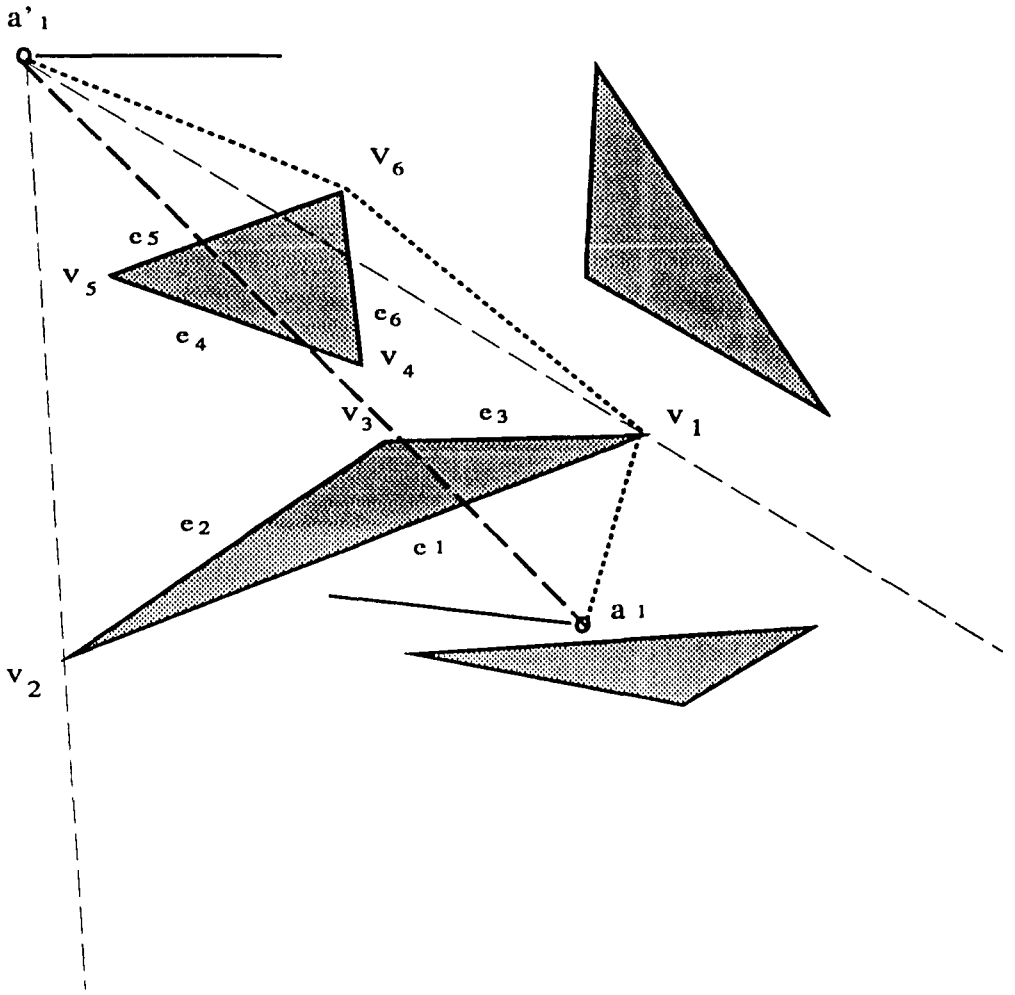


Figure 3.2: A simple example of an edge blocking a path.

the motion that passes through the extension of $\overline{a_1' v_1}$ ($\overline{a_1' v_2}$) to as *clearing* the vertex v_1 (v_2). Had we chosen to clear v_1, any subsequent movements should clear either v_4 or v_5 to avoid the edge e_4. Suppose v_1 and v_4 are chosen as the vertices to be cleared. If we do not consider the other part of the ladder, the shortest path for A_1 to clear both vertices is to follow the straight line between v_1 and v_4. As the ladder is a rigid object, the other part of the ladder also has to clear both v_1 and v_4 in order for A_1 to clear v_1 and v_4. Here, the line segment $\overline{v_1 v_4}$ is said to be a *guide* line for this particular pair of the origin and destination, a_1 and a_1'.

Literally speaking, any visible pair of obstacle vertices can form a guide line for a

certain combination of the origin and destination. However, some of the guide lines are not available as certain ladder positions lining up with the guide lines are not free of obstacles. Also, the ladder may not move from one guide line to another due to the surrounding obstacles. For example, in Fig. 3.2, $\overline{v_3v_4}$ is not valid a guide line because the short distance between the two vertices forbids for the ladder to line up with $\overline{v_3v_4}$. On the other hand, the ladder lined up with $\overline{v_3v_5}$ needs non-trivial motion to move to a position that lines up with $\overline{v_3v_6}$. The Reachability Graph (RG) is based on the graph representing the availability of such guide lines and their connectivity. More precisely, the RG is defined as follows.

Definition 1 *The RG is a graph* $\{N, E\}$ *such that*

$$N \subset I_2 \times V^2, E \subset N \times N$$

where $I_2 = \{1, 2\}$ *is the set of the ladder's two ends and* V *is the set of obstacle vertices. There is a node* $n = \{i, v_j, v_k\} \in N$ *when*

1. v_j *is within the line of sight (i.e., visible) from* v_k,
2. *if* $i = 0$, $c = \{v_j, \lambda_{v_j,v_k}\}$ *is a free position of the ladder where* λ_{v_j,v_k} *is the orientation of the line connecting* v_j *and* v_k, *or*
3. *if* $i = 2$, $d^*(\{v_j, \lambda_{v_j,v_k}\})$ *is a free position of the ladder.*

There is an edge between the two nodes $n_1 = \{i, v_j, v_k\}$ *and* $n_2 = \{i', v'_j, v'_k\}$ *when*

1. $i \neq i', v_j = v'_k$ *and* $v_k = v'_j$, *or*
2. *either* $v_j = v'_j$ *or* $v_k = v'_k$ *and there is a path for the ladder to follow while staying on the obstacle vertex* v_j.□

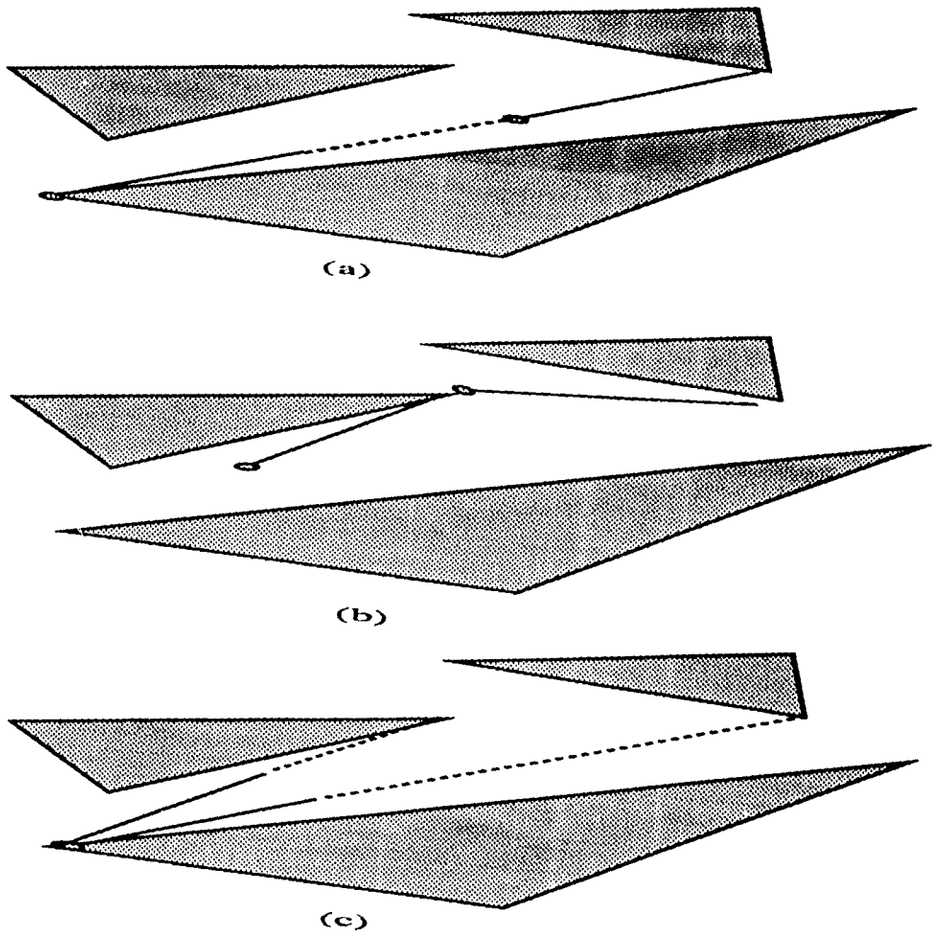


Figure 3.3: Three different types of connections between two nodes.

A node of the RG represents a position of the ladder at the guide line formed by two visible obstacle vertices, provided the position is free from obstacles.

There are three types of edges in the RG (see Fig. 3.3). They are:

- Edges corresponding to straight line paths along the guide line. (Type I edge in Fig. 3.3(a).)
- Edges corresponding to sliding motion paths between two positions in the opposite sides of a vertex on which the line segments are located. (Type II edge in Fig. 3.3(b).)
- Edges corresponding to rotational paths between two neighboring guide lines sharing an obstacle vertex. (Type III edge in Fig. 3.3(c).)

By the definition of the RG, each node has at most 5 edges. They are: one type I edge, two type II edges and two type III edges. The existence of a type I edge is guaranteed by the mere existence of the two nodes connected by the edge. However, type II and type III edges may or may not exist depending on the surrounding obstacles. In the next section, we will describe the factors that determine nodes and their edges to form a reachability graph.

3.3 Construction of a Reachability Graph

All vertices are assumed to be doubly-linked with those vertices with which they form obstacle edges. Since polygons are assumed to be simple polygons, each vertex is linked with two other vertices. For convenience of representation, we denote the two vertices that form obstacle edges with a vertex v as $cw(v)$ and $ccw(v)$.

First, we need to obtain the set of all visible edges from each obstacle vertex. In some cases, only part of an edge is visible from the vertex. In other cases, none of

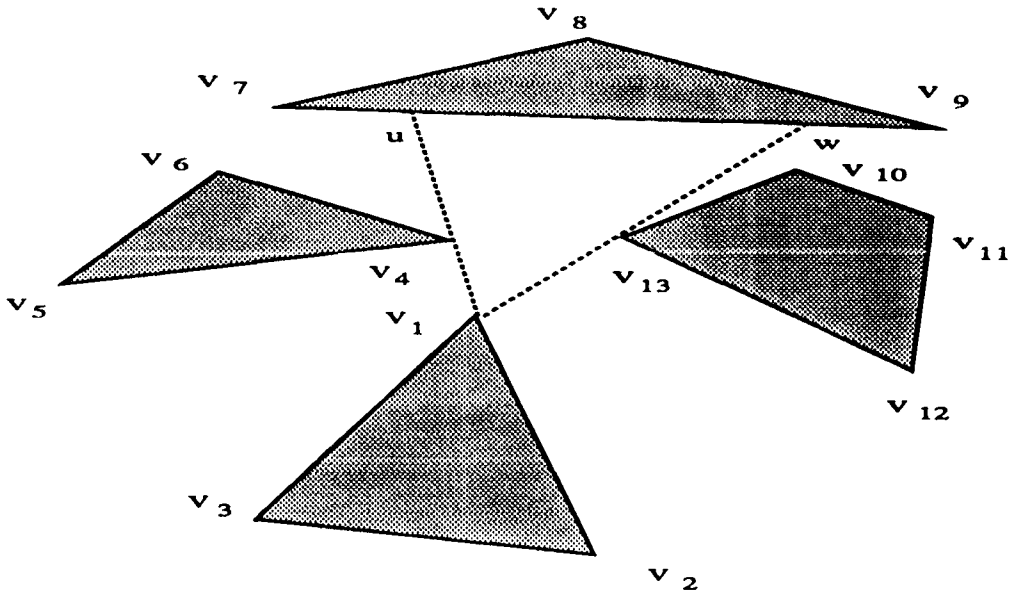


Figure 3.4: An example of background edges.

the vertices of the visible edge is visible from the vertex due to the overlapping with next visible edges. Therefore, we not only have to obtain all pairs of visible vertices but also have to get all pairs of edges which are partially visible. We will call such edges *background edges*. In the example of Fig. 3.4, the edge $\overline{v_7v_9}$ is a background edge from v_1's standpoint as neither v_7 nor v_9 is visible from v_1 while part of this edge, $\overline{uw}$, is visible from v. Node $\{1, v_1, v_4\}$ may or may not exist depending on the length of the ladder and the distance between v_1 and u. Same can be said between v_1 and w. We need to know all the background edges to determine whether or not any particular position is free.

As there already exist many known algorithms to obtain such visible edges, we will use one of them. One may find more detailed algorithms and a good survey of such algorithms in [75]. We assume that such an algorithm returns a doubly-linked list of visible edges, **Visible_Edge_List**, for each given vertex. The list is sorted by polar angle using v_1 as the origin of the polar coordinates.

Using a vertex v's list of visible edges, we now compute the distance between each visible edge and the vertex v. Let $\overline{p_i p_j}$ be a visible edge from v and $\delta_{v,\overline{p_i p_j}}(\theta)$ be the distance between the visible edge $\overline{p_i p_j}$ and v at angle θ. Then, $\delta_{v,\overline{p_i p_j}}(\theta)$ can be computed using

$$\delta_{v,\overline{p_i p_j}}(\theta) = \begin{cases} \frac{d(\overline{p_i p_j}, v)}{\sin(\theta - \lambda_{p_i,p_j})} & \theta \in [\lambda_{v,p_i}, \lambda_{v,p_j}] \\ \infty & \text{otherwise.} \end{cases}$$

Then, the distance to any visible edge from v at angle θ becomes

$$\delta_v(\theta) = \min_{\overline{p_i p_j} \in L} \delta_{v,\overline{p_i p_j}}$$

where L is the **Visible_Edge_List**.

Using this distance function $\delta_v(\theta)$, we can now determine the existence of a node in the RG as follows.

Theorem 1 *There exists a node $n = \{1, v, v_i\}$ in the RG if and only if*

$$d(v, v_i) \leq \delta_v(\lambda_{v,v_i}), d(A_1, A_2) \leq \delta_v(\lambda_{v,v_i})$$

where $d(v, v_i)$ is the distance between v and v_i.

Proof: The condition $d(v, v_i) \leq \delta_v(\lambda_{v,v_i})$ is the definition of a visible vertex while the second condition defines the position corresponding to the node that represents a free space. By the definition of a RG, the node n and its dual $n* = \{1, v_i, v\}$ are valid members of the node set of the RG. ∎

Corollary 1 *If there exists a node $n = \{1, v, v_i\}$ in the RG, there also exists a node $n^* = \{2, v, v_i\}$ in the RG.*

One can always determine whether or not there exists an edge between two nodes as in the definition of RG. As mentioned in the previous section, there are three types

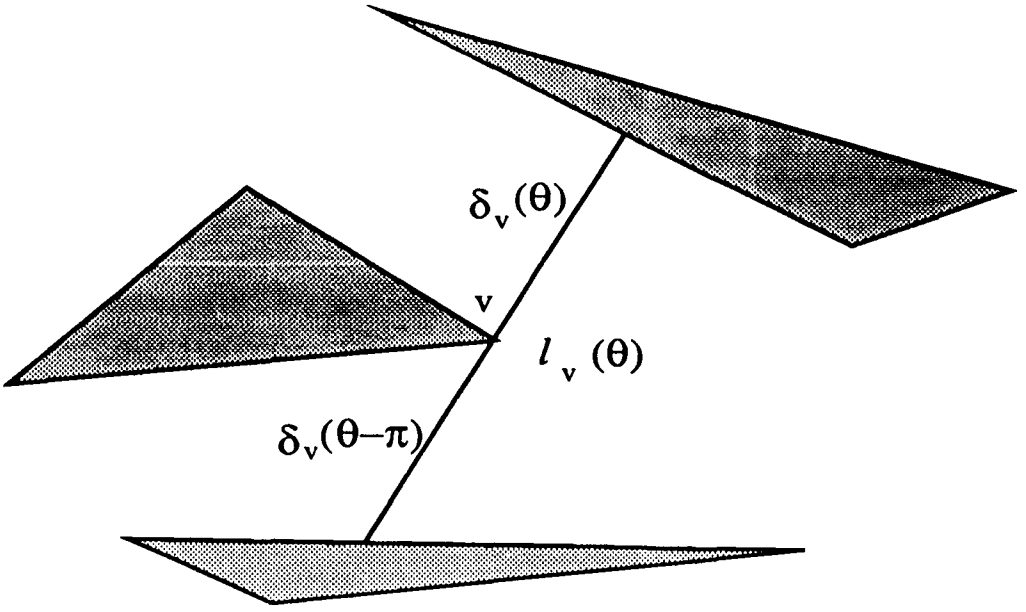


Figure 3.5: The maximum allowable length of the line segment on a vertex.

of edges. The existence of Type I edges is guaranteed by the existence of the nodes connected by the edges. Type II and Type III edges can be obtained using $\delta_v(\theta)$. According to the definition of RG, an edge between two nodes exists when there is a continuous path between the two positions represented by the nodes while the ladder remains on v. This means that there should be a free space available for the ladder lying at all angles between its orientations at the two positions. One can determine whether such positions exist or not by checking the following function. Let $l_v(\theta)$ be the maximum length of a line segment on vertex v at angle θ (see Fig. 3.5). Then, $l_v(\theta)$ becomes

$$l_v(\theta) = \delta_v(\theta) + \delta_v(\theta - \pi).$$

The condition for the existence of a continuous path of the ladder on v at an angle in $[\theta_1, \theta_2]$ is

$$\inf_{\theta \in [\theta_1, \theta_2]} l_v(\theta) \leq | \overline{A_1 A_2} | .$$

Fig. 3.6 is the plot of $\delta_v(\theta)$ and $l_v(\theta)$ for the example in Fig. 3.4. Since $l_v(\theta)$ as well

as $\delta_v(\theta)$ is a combination of several functions, the infimum may occur at the infimum of each function within its range. The infimum of $\delta_{v,\overline{v_iv_j}}(\theta)$ occurs only at one of the following orientations

$$\begin{aligned}\theta &= \lambda_{v,v_i} \text{ or} \\ &= \lambda_{v,v_j} \text{ or} \\ &= \lambda_{v_i,v_j} + \pi \text{ if } (\lambda_{v_iv_j} + \pi) \in [\lambda_{v,v_j}, \lambda_{v_i,v_j}].\end{aligned}$$

We denote the above three angles as $\Lambda_{\overline{v_iv_j}}(v)$. Then, the connection between $\{1, v, v_i\}$ and $\{1, v, v_j\}$ exists when

$$l_v(\theta) \leq |\ \overline{A_1A_2}\ | \text{ for } \theta \in \Lambda_{\overline{v_iv_j}}(v) \bigcup_{\overline{v_kv_l} \in \Xi} \Lambda_{\overline{v_kv_l}}(v)$$

where Ξ is the set of all edges in the opposite side of $\overline{v_iv_j}$. From now on, we refer $\Lambda_{\overline{v_iv_j}}(v)$ to be as the set of critical angles of $\overline{v_iv_j}$ from v's standpoint.

Using the set of critical angles, we now describe our algorithm **Scan** that actually constructs the RG. For each obstacle vertex, **Scan** computes all the nodes of the RG by obtaining all the guidelines generated by the obstacle vertex. This can be achieved by checking all possible guide lines that can be generated by the vertex. That is, **Scan** must check all the obstacle vertices visible from the vertex and determine whether the line connecting the two vertices is a valid guide line or not.

Suppose v_i is the obstacle vertex **Scan** is currently working on. First, **Scan** needs to compute the set of all visible obstacle vertices. This has been studied extensively by several researchers [43, 75, 1], so their results will be used by **Scan.** Let **Visible_Edge_List** of v_i be the set of all obstacle vertices visible from v_i, sorted by their polar angle with v_i as their reference point.

Next, the algorithm scans **Visible_Edge_List** until it finds the edge that is on the opposite side of the first vertex of the list in v_i's view (see Fig. 3.7). This is done

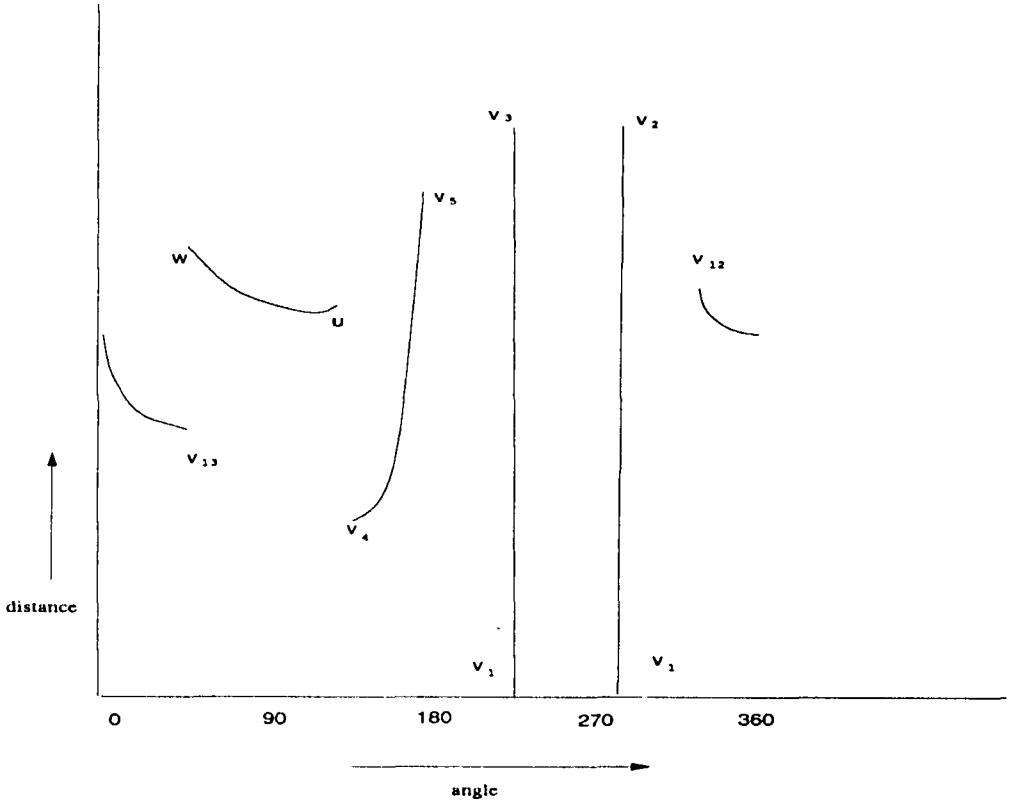


Figure 3.6: Distance function of v.

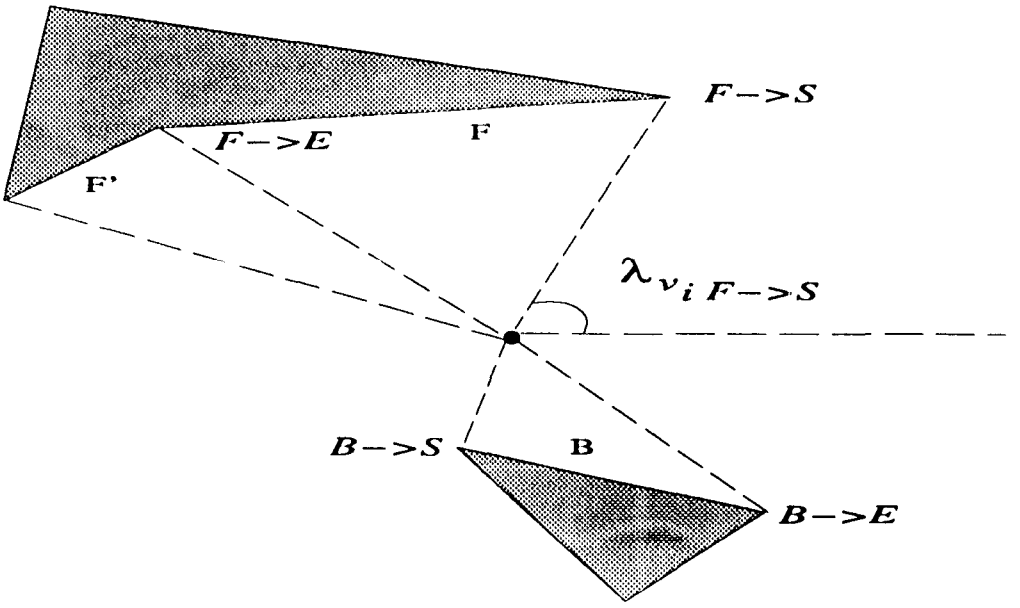


Figure 3.7: Two edges in the opposite direction of an obstacle vertex.

in step 2 and step 3 of **Scan**. In step 2, the two pointers F and B are initialized to point to the first edge in **Visible_Edge_List**. In step 3, **Scan** advances F to point to the next edge in **Visible_Edge_List** while leaving B to the first edge until F is opposite direction of B in v_i's view.

Then, **Scan** first determines whether or not a particular visible line is a valid guide line or not by comparing the length of the ladder, $|\overline{A_1A_2}|$, with the distance, $l_{v_i}(\theta)$, between the two edges F and B, where θ is the set of critical angles due to F and B. If it is a valid guide line, **Scan** generates the corresponding nodes of the RG and connects them accordingly. Notice that two nodes are generated so that **Scan** can accommodate the dual of each ladder position. This procedure continues while advancing F and B alternately until B points to the first edge of **Visible_Edge_List** again.

Procedure **Scan**(v_i)

1. Sort the element of **Visible_Edge_List** of v_i by their polar angles.

2. Assign F and B to the first edge of **Visible_Edge_List**.

3. **While** $\lambda_{v_i F \to S} - \lambda_{v_i B \to E} < \pi$ **do** F := RLINK(F) **end**

4. **While** RLINK(F) $\neq$ START

(4a) **if** $\lambda_{v_i F \to S} > \lambda_{v_i B \to E} + \pi$ **then**

if $l_{v_i}(\theta) > |\overline{A_1 A_2}|$ for $\theta \in \Lambda_F \cap \Lambda_B$

Determine visible vertices w and x for $F \to S$ and $F \to E$.

Connect $\{1, v_i, w\}$ and $\{1, v_i, x\}$.

Connect $\{2, v_i, w\}$ and $\{2, v_i, x\}$.

(4b) **else**

if $l_{v_i}(\theta) > |\overline{A_1 A_2}|$ for $\theta \in \Lambda_F \cap \Lambda_B$

Determine visible vertices w and x for $F \to S$ and $B \to S$.

Connect $\{1, v_i, w\}$ and $\{2, v_i, x\}$.

Connect $\{1, v_i, w\}$ and $\{2, v_i, x\}$.

(4c) **if** $\lambda_{v_i F \to S} > \lambda_{v_i B \to S} + \pi$ **then** F := RLINK(F)

(4d) **else** B := RLINK(B).

Now, we describe some of the characteristics of the algorithm **Scan** and the graph, RG, it generates.

Theorem 2 *The computational complexity of the algorithm* **Scan** *is* $O(n \log n)$ *where* n *is the total number of edges in the workspace.*

Proof : Each of the steps in the algorithm **Scan** involves only comparisons and arithmetic operations. All steps except for Step 1 are the linear processing of an n-element list. In Step 1, a list of length n needs to be sorted. Hence, the time complexity of Step 1 is $O(n \log n)$. In Step 4, we need to check at most n edges, and there are at most 6 positions to check before determining the connectivity. Therefore,

the complexity of Step 4 is $O(n)$. So, the overall complexity is $O(n \log n)$. ∎

Since there are n vertices, we need to call **Scan** n times. Therefore, the complexity of the overall algorithm is $O(n^2 \log n)$.

Let's consider the example workspace in Fig. 3.9. A part of the reachability graph generated for this example is shown in Fig. 3.8. Due to the complexity of the graph, the figure shows only those nodes related to b_2 and b_5. In this example, there is no Type III edge involving b_2 and b_5 in the RG. Basically, Type I and Type II edges always generate two symmetric nodes corresponding to the dual of the ladder position. This results in two identical subgraphs in the RG. Depending on the arrangement of the obstacles, these two subgraphs may or may not be connected. However, a Type III edge always connects two nodes, one from each subgraph. Without any Type III edge, the RG shown in Fig. 3.8 is not connected, implying the existence of certain origin and destination pairs that cannot be connected.

Suppose a specific origin and destination pair is given and we want to move a ladder from $\{a, b\}$ to $\{c, d\}$ in the example workspace of Fig. 3.9. Using the algorithm described in Section 3, we can construct the RG. In addition to the graph obtained from the obstacles, we add four temporary vertices to the workspace as shown in Fig. 3.4. Basically, these four vertices are the positions of A_1 and A_2 at the origin and those at the destination. In this example, we added temporary vertices a, b, c, and d. After adding these vertices, we can compute the added nodes of the RG using the algorithm **Scan**. Upon construction of the RG, we may use any graph search algorithm to determine the path between the origin and destination.

Corollary 2 *The addition of temporary vertices can be computed in* $O(n \log n)$ *time.*

Proof: Since each additional vertex requires $O(n \log n)$ time complexity, adding four

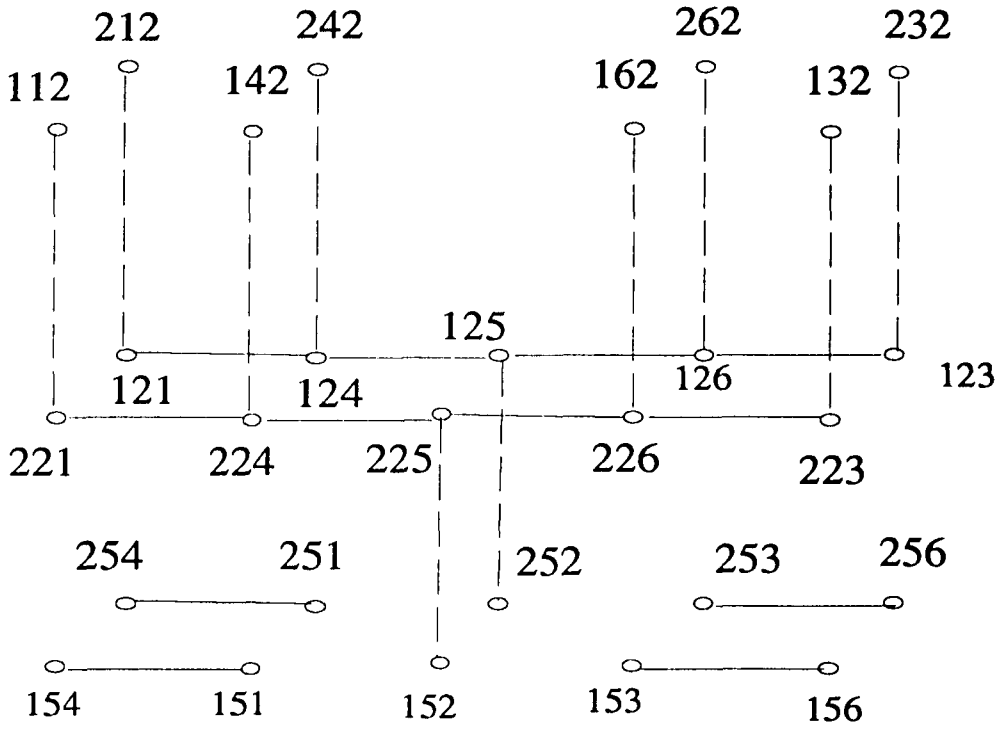


Figure 3.8: A sample RG with the addition of the origin and the destination.

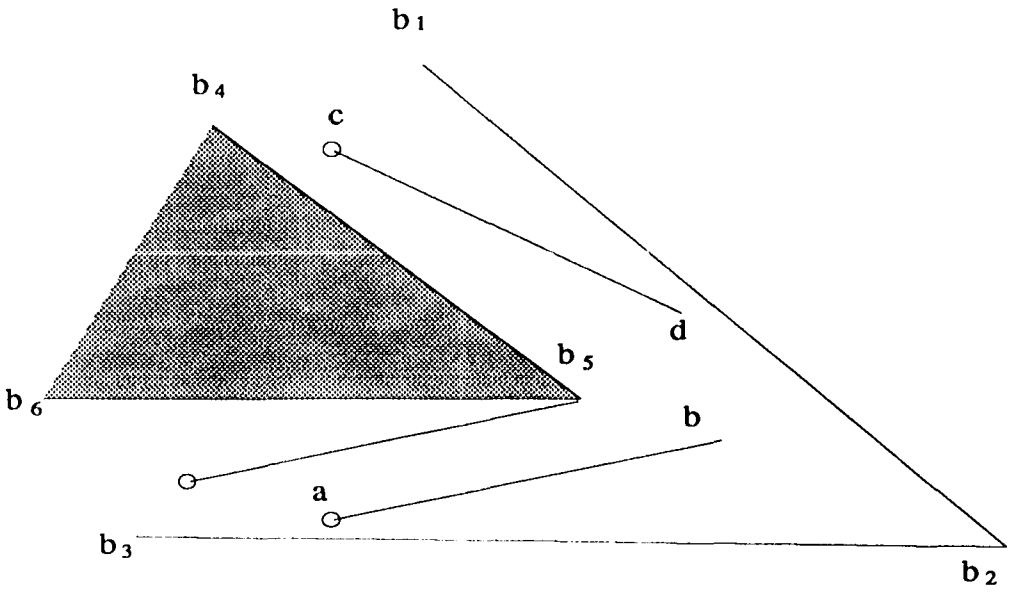


Figure 3.9: An example with origin and destination.

more vertices can be done with additional $O(n \log n)$ time complexity. Therefore, overall complexity remains to be $O(n^2 \log n)$. ∎

After adding the origin and destination to the graph, we can obtain a path between any origin and destination in $O(n^2 \log n)$ time.

In the following section, we will prove the correctness of our algorithm and discuss the minimum complexity bound of the ladder problem in general.

3.4 The Minimum Complexity Bound of a Path Planner

Let's consider a ladder at some position c and its straight-line extension, denoted by $e(c)$. The position c is in free space when $e(c)$ intersects obstacle edges outside the ladder (see Fig. 3.10). Using this observation, Leven and Sharir labeled a free space with $s(c) = \{e_i, e_j\}$ where e_i and e_j are two obstacle edges closest to the ladder among those obstacle edges that intersect the extension of the ladder. Let P_i be the

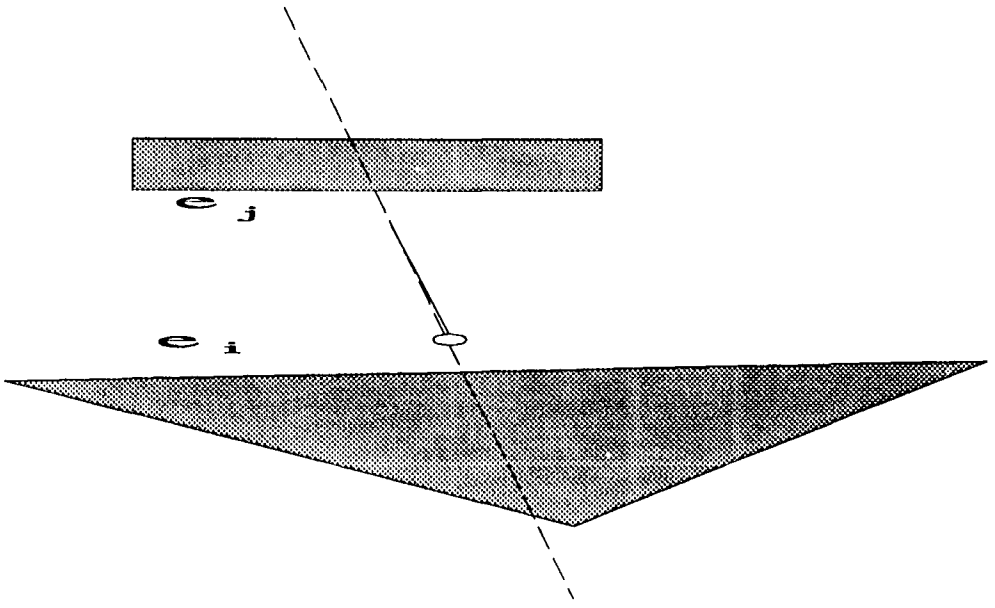


Figure 3.10: A position of the ladder and its extension.

set of all free spaces such that

$$P_i = \{c : s(c) = \{e_i, e_j\} \text{ for some } e_j\}.$$

Obviously, any two obstacle edges e_i and e_j can form a region of free space labeled with $\{e_i, e_j\}$ as long as some part of e_j is within the line of sight from a part of e_i. Then, P_i can be represented by a Planar Straight Line Graph (PSLG) which consists of obstacle edges. In the example shown in Fig. 3.11, P_1 is represented by $e_2e_3e_6e_5e_7e_3e_4$.

A PSLG forming P_i for some obstacle edge e_i is characterized by the following lemma.

Lemma 1 *Let e_i be an obstacle edge. Then, the length of the PSLG formed by P_i is $\Omega(n)$ and P_i can be any one of $\Omega(n^n)$ possible PSLGs.*

Proof: The proof of the length of the PSLG is straightforward and left to readers. One extreme example is shown in Fig. 3.11, where P_1 contains all edges in the

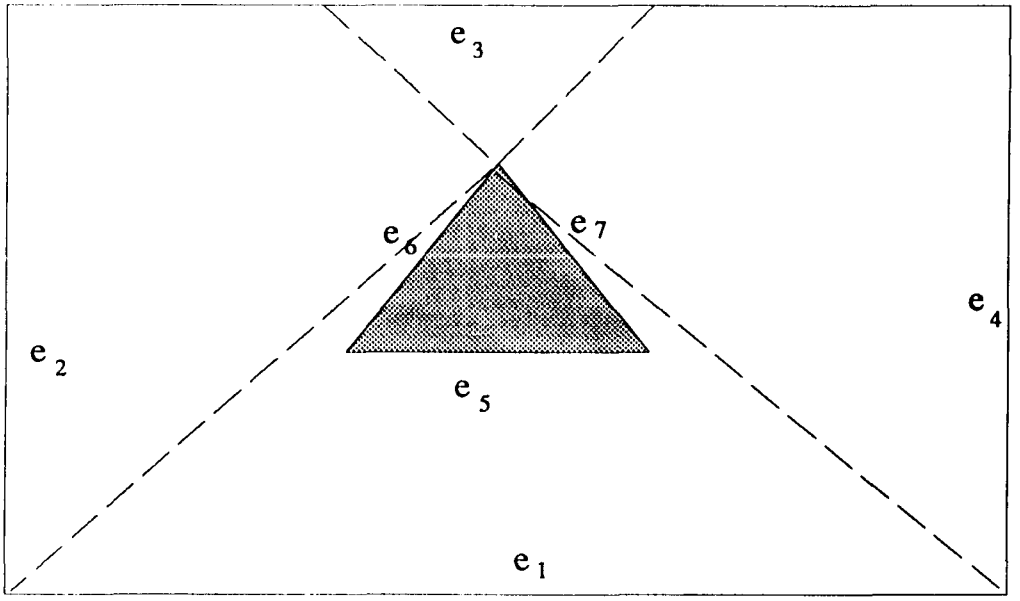


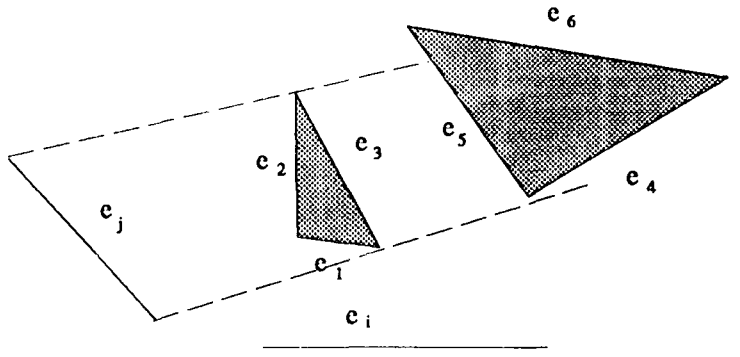
Figure 3.11: The region formed by a PSLG generated by an edge.

workspace except e_1 itself. Furthermore, there is a certain edge, e_3, which appeared more than once in the sequence. The total number of possible PSLGs corresponds to the number of possible words whose length is $\Omega(n)$ over the alphabet of size n. Thus, there are $\Omega(n^n)$ different PSLGs that an obstacle edge can be associated with. ∎

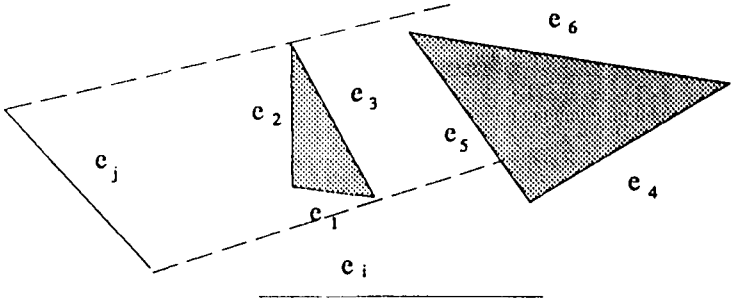
Next, we will show the total number of different PSLGs that a given workspace can have.

Lemma 2 *Let e_i and e_j two obstacle edges in a workspace then their PSLGs, P_i and P_j, may or may not contain any common edge and the order of common edges, if any, in P_i does not affect that in P_j.*

Proof: We will show two workspaces that are almost identical but have different PSLGs. In the examples in Fig. 3.12(a) and Fig. 3.12(b), P_i contains a substring $e_2e_1e_3e_5e_4$. However, P_j contains a substring $e_5e_2e_1$ for the example in Fig. 3.12(a) and $e_2e_1e_5$ for that in Fig. 3.12(b). That is, the positions of e_5 in the PSLG codings



(a)



(b)

Figure 3.12: Two otherwise identical workspaces with two different codings.

are different for otherwise identical workspaces. Also, note that e_5 may not even appear in P_j at all, had the length of e_5 been a little shorter. This proves that the PSLG of one edge does not affect those of other edges. ∎

Using Lemmas 1 and 2, we now prove the worst-case computational complexity of the ladder problem.

Theorem 3 *There are $\Omega(n^{n^2})$ possible PSLGs for a workspace consisting of n obstacle edges and any non-heuristic path planning algorithm requires $\Omega(n^2 \log n)$ steps.*

Proof: According to Lemmas 1 and 2, there are n independent PSLGs each with $\Omega(n^n)$ possible combinations. Hence, there are $\Omega(n^n) \times \Omega(n^n) \times \ldots \times \Omega(n^n)$ PSLGs. That is, there are a total of $\Omega(n^{n^2})$ different PSLGs with which a workspace can be represented. Suppose there is a non-heuristic path planner that generates a path using the description of workspace, origin, and destination. Being non-heuristic, the planner should always find a path if any. Consider the use of the path planner for two workspaces with two different PSLG codings, $P = P_1 P_2 \cdots P_n$ and $P' = P'_1 P'_2 \cdots P'_n$. Suppose P and P' are identical except P_k and P'_k where P_k has an edge e_l while P'_k does not. Any non-heuristic path planner should recognize the difference between P and P' or it will generate identical paths for both workspaces. Without knowing whether the ladder position with label $\{e_k, e_l\}$ is free or not, the path planner has to always avoid such a position. However, it is not possible to entirely avoid such positions as a certain pair of origin and destination may require the path to include some of them (e.g., the origin or the destination may start from such a position). To differentiate all the workspaces with different PSLGs, the planner requires at least $\log_c \Omega(n^{n^2})$ comparison steps, provided it can make only c comparisons at a time. By taking the logarithm inside of the arguments of Ω, the planner needs at least $\Omega(\log(n^{n^2})) = \Omega(n^2 \log n)$ comparisons. Hence, the ladder problem requires $\Omega(n^2 \log n)$ complexity. ∎

Now, we will prove the correctness of the proposed algorithm by showing that the RG has the same power in representing the workspace as the connectivity graph (CG) used in [40].

Lemma 3 *Let $G = \{V, E\}$ be the RG and $G' = \{V', E'\}$ be the CG whose nodes consist of the trapezoids described in [40]. Then, for every $v \in V$, there exists a $v' \in V'$ such that the ladder position corresponding to v is one of the parallel edges*

of the trapezoid represented by v'.

The proof is straightforward as both the guide lines used in the RG and one of the parallel edges of each trapezoid of the CG are generated by the visibility lines connecting two obstacle vertices. The converse of the lemma is not true as there are trapezoids in the CG which do not have corresponding guide lines in the RG. The effects of such trapezoids without the corresponding guidelines can be described by the following lemma.

Lemma 4 *Let D be the difference between the vertex set of the RG and the CG. Then, for every $d \in D$ and any path of the ladder that passes through the trapezoid corresponding to d, there is an alternative path of the ladder that passes through another trapezoid that has a corresponding guide line in the CG.*

Proof: **Scan** does not generate any node when the distance between two obstacle vertices are shorter than the length of the ladder. For example, in Fig. 3.13, **Scan** does not generate any node corresponding to e_l because e_l is shorter than the ladder. By contrast, the CG should contain a node corresponding to e_l as the length of e_m can be longer than that of the ladder. However, any path that passes through the trapezoid described by e_l and e_m can be represented by the guide line represented by e_n. ∎

Based on Lemmas 3 and 4, we can conclude that the RG has the same power as the CG in representing the workspace.

3.5 Summary

One of the main advantages of using the RG over the CG is that the size of the RG is much smaller than that of the CG. Hence, the actual search time can

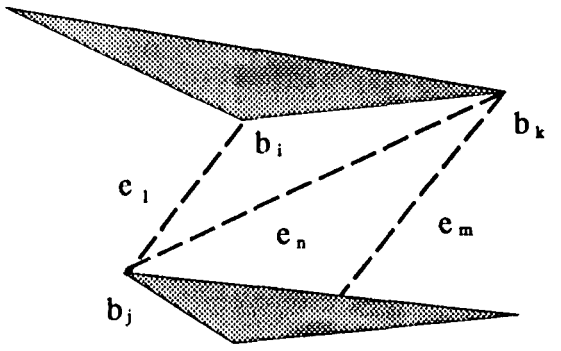


Figure 3.13: Effect of short guide line.

be reduced. This is particularly true when the workspace is crowded with obstacles separated by short distances.

As the computational complexity of the RG is the same as the theoretical minimum of the general ladder problem, using the RG is an computationally optimal approach to our ladder problem. Note that there may be more efficient algorithms under some additional restrictions to the obstacles. For example, we may find a more efficient algorithm for a workspace where the distances between the closest obstacle edges exceed a certain value. Though these restrictions may be too severe for its general applicability, such an algorithm may fare very well in actual implementations.

CHAPTER IV

DOMINANCE GRAPH AND ITS APPLICATIONS

4.1 Introduction

The most popular solution to the shortest path planning problem (SPPP) hinges on the visibility graph (VG). The VG method is based on the observation that when two points in a plane are not visible from each other, the shortest path always contain one or more vertices of the obstacle in the plane. Based on this observation, the workspace is transformed into a graph in which the distances between all pairs of mutually visible vertices are precalculated. The optimal solution can then be obtained using the Dijkstra's graph search algorithm [15].

Though the VG is very useful in 2D, it is very difficult to use in higher dimensional spaces. For example, when two points are not visible from each other in 3D, the shortest path between them should pass through one or more edges of the obstacles instead of their vertices. Consequently, it is difficult to represent a 3D workspace with a graph. Moreover, unlike the case of vertices, the shortest paths passing through edges are not unique. For example, in 2D the length of the shortest path between two points, A and B, passing through a third point C is $\overline{AC} + \overline{CB}$. In 3D, the length of the shortest path between the two points passing through an edge c is not

usually the sum of the distance between A and c and that between B and c, implying the difficulty in using Dijkstra's graph search algorithm. This difficulty has led to the development of heuristic, application-specific approaches, such as the ones in [31, 23].

We shall introduce the L_1 *visibility* between two points in a digitized workspace, based on which we can derive dominance relations between certain partitions of the workspace. These dominance relations show some useful properties that can be utilized to solve the SPPP.

The chapter is organized as follows. Section 4.2 states the SPPP formally. In Section 4.3 we define L_1 visiblity and demonstrate how it partitions the workspace. In Section 4.3.1, the properties of a partitioned workspace are examined. Section 4.3.2 presents a graph representation of the workspace based on which an SPP solution algorithm is derived. Section 4.4 presents an example and simuilation results. The chapter concludes with Section 4.5.

4.2 Problem Statement

Consider the problem of moving an object in a workspace cluttered with obstacles. We want to find a path, or determine a set of points, for the object to traverse from a starting point (origin) to an end point (destination) without colliding with any obstacle in the workspace. There are two sources of difficulty associated with this problem: (i) an infinite number of paths exist for each given origin-destination pair, and (ii) it is in general difficult to represent obstacles of arbitrary shape in the workspace. One way of circumventing these sources of difficulty is to divide the workspace into a finite number of cells[1]. Such division not only reduces the infinite

[1] A cell is a square in 2D and a cube in 3D.

number of possible paths to a finite number of paths, but also allows each obstacle to be represented by the set of cells it occupies.

Let a three dimensional workspace be divided into $\ell \times m \times n$ identical cells. According to the CSA [44], the object to be moved can be shrunk to a point by growing obstacles. In what follows, cells are represented as $o, p, q, \ldots$ when their locations need not be specified, as $o_{ijk}, p_{\ell mn}, q_{abc}, \ldots$ when their locations need to be specified, and as $v_1, v_2, v_3, \ldots$ when a sequence of cells needs to be specified. Informally, the goal of a path planner is to find a path formed by a sequence of neighboring free (unoccupied) cells from the origin to the destination while minimizing a certain path cost.

The most commonly used cost is path length. In a Euclidean space E^d, the L_p-distance, $d_p(x, y)$, between two points, $x = (x_1, x_2, \ldots, x_d)$ and $y = (y_1, y_2, \ldots, y_d)$, is defined as:

$$d_p(x,y) = (\mid x_1 - y_1 \mid^p + \mid x_2 - y_2 \mid^p + \ldots + \mid x_d - y_d \mid^p)^{1/p}, \text{ where } 1 \leq p < \infty$$

$$d_\infty(x,y) = \max(\mid x_1 - y_1 \mid, \mid x_2 - y_2 \mid, \ldots, \mid x_d - y_d \mid).$$

Though there exist an infinite number of L_p-metrics, only three of them have significance for path planning: L_1, L_2, and L_∞. The advantage of using L_2-metric over L_1- or L_∞- metric is its ability of describing the object's traversal distance. On the other hand, L_1- and L_∞- metrics have the following advantages:

- The paths generated under L_1- or L_∞- metric are usually safer[2] than those under L_2-metric.

- L_∞-metric (or L_1-metric) contains more accurate information concerning the minimum number of search steps required from the current location to the

[2]By 'safer', we mean that the minimum and average clearance from obstacles are greater.

destination than any other metrics.

- The length of a path in a digitized space is the same as the path in the corresponding continuous space when L_1- or L_∞- metric is used.

We will limit our discussion to L_1-metric only, since, as shown in [37], any problem in L_1-metric space can be transformed into an equivalent problem in L_∞-metric space with a simple change of coordinate system. The *cost* of a path P, denoted by $C(P)$, is the length of P measured in L_1-metric. Two cells are said to be *neighbors* if they are physically adjacent. Since all the cells are identical in size, the L_1-distance between the centers of any two neighboring cells are identical and will be treated as *unit distance*. (i.e., Two cells a and b are adjacent if $d_1(a,b) = 1$.

The path planning problem can now be stated formally as follows: For given two points, x and y, and a set, $\mathcal{O}$, of cells that are occupied by obstacles, find a sequence, $P = v_1 v_2 \ldots v_n$, of neighboring cells such that

$$x \in v_1,\ y \in v_n,\ d_1(v_i, v_{i+1}) = 1, v_i \notin \mathcal{O} \text{ for } 1 \leq i \leq n-1,$$

while minimizing n.

4.3 Partitioning the Workspace

It is necessary to define the following terms for the clarity of presentation.

Definition 2 *In L_1-metric space, a cell v is said to be* visible *from a cell w, denoted by $v \Re w$, if there exists a sequence $P = v_0 v_1 \ldots v_{d_1(v,w)}$ of free cells such that*

$$v_0 = v,\ v_{d_1(v,w)} = w,\ d_1(v_{i-1}, v_i) = 1,\ \textit{and}\ d_1(v_i, v_{d_1(v,w)}) = d_1(v,w) - i,\ 1 \leq i \leq d_1(v,w),$$

where $d_1(u,v)$ is the L_1-distance between u and v. Otherwise, v is said to be not visible *from w, denoted by $v \not\Re w$.*

Definition 3 *For any two adjacent cells, a set,* $N = \{n_x^+, n_x^-, n_y^+, n_y^-, n_z^+, n_z^-\}$, *is called the set of* neighbor operators *if*

$$\begin{aligned} n_x^+(v_{ijk}) = v_{\ell mn} &\implies \ell = i+1,\ m = j,\ n = k \\ n_x^-(v_{ijk}) = v_{\ell mn} &\implies \ell = i-1,\ m = j,\ n = k \\ &\vdots \\ n_z^-(v_{ijk}) = v_{\ell mn} &\implies \ell = i,\ m = j,\ n = k-1. \end{aligned}$$

and a set, $O \subset N$, *is called the* orthogonal *set of neighbor operators if they always generate the neighbors in orthogonal directions of a cell, e.g.,* $\{n_x^+, n_y^-, n_z^+\}$.

Definition 4 *For any two free*[3] *cells,* v *and* w, *in the workspace,* v *is said to be* visible *from* w *if there exists an orthogonal set,* O, *of neighbor operators such that*

$$P = v_0 v_1 \ldots v_{d_1(v,w)} \text{ where } v_0 = v,\ v_{d_1(v,w)} = w,\ v_i = n(v_{i-1}) \text{ for some } n \in O,$$

and O *is called the* generating operator set *of* P. *The dual of* O, *denoted by* O^*, *is the generating operator set of the reverse sequence of* P, *i.e.,* $v_{d_1(v,w)} v_{d_1(v,w)-1} \ldots v_0$.

Using the above definition of visibility, the *dominance* relation between cells and that between two sets of cells are defined as follows.

Definition 5 *For any two cells* u *and* v *in a workspace* W, u *is said to dominate* v, *denoted by* $u >_c v$, *iff* $w \Re v \rightarrow w \Re u, \quad \forall\ w \in W$. *Similarly, for any two sets,* A *and* B, *of cells in* W, A *is said to dominate* B, *denoted by* $A >_s B$, *iff for any* $v \in B$, $\exists u \in A$ *such that* $u >_c v$.

Having defined the dominance relation between cells, the *equal* relation is defined as follows.

[3] A cell is said to be *free* if it does not intersect any obstacle.

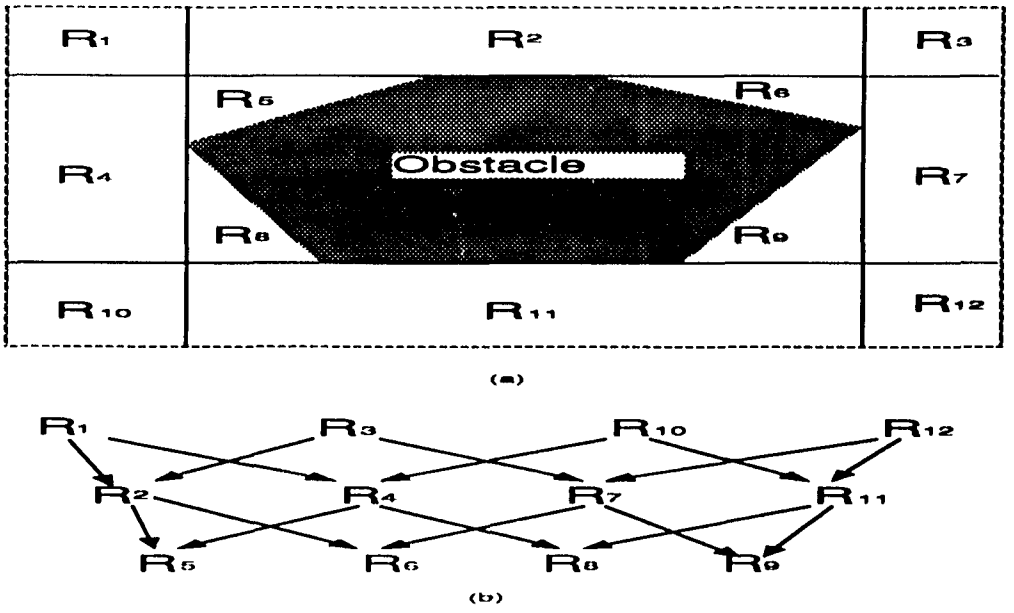


Figure 4.1: Partitioning of the workspace into regions.

Definition 6 *For any two cells u and v, u is said to be* equal *to v, denoted by $u \sim_c v$, iff $u >_c v$ and $v >_c u$.*

Notice that the relation $\sim_c$ is an equivalence relation[4] and its central importance is that it induces a partition of the workspace. That is, the relation $\sim_c$ divides the workspace into several sets of cells such that $u \in R(v) \rightarrow S(u) = S(v)$, where $R(v)$ is a partition containing v and $S(u)$ the set of all the cells that are not visible from u. Such a set will henceforth be referred to as a *region*. In the 2D example of Fig. 4.1, any cell in R_1 is visible from any other cell in the entire workspace except for those in R_9. Similarly, no cell in R_2 is visible from any cell in $R_8 \bigcup R_9 \bigcup R_{11}$.

There are several ways of obtaining regions. In case of 2D space, a border of the regions is formed by projecting the edges of obstacles along x and y directions, as shown in Fig. 4.1. The following procedure, **P1**, is used to determine the total number of cells visible from a given cell. Since total numbers of cells visible from

[4]It is reflexive, symmetric, and transitive.

two equivalent cells are same, **P1** can be used to partition the workspace under the assumption that no two neighboring regions have the same number of visible cells. (This assumption can be relaxed trivially as we shall see shortly.) Informally, the procedure works as follows. All the cells visible from a given cell, v, of the workspace are obtained and expressed with an indicator vector I, i.e., $I(x) = 1$ (0) if a cell x is free (occupied by an obstacle). Starting with a cell v of distance 0 (i.e., itself), one can determine all the visible cells of distance 1 from v. Using a recursion, one can then calculate all the cells visible from v of distance $2, 3, \ldots K$, where K is the maximum possible distance between any two cells in the workspace. In most cases, K is three times the resolution of each axis. After determining all the cells visible from a given cell v, the total number, $N(v)$, of cells visible from v is obtained from the vector I.

Procedure **P1**

For every cell v in the workspace W

if v is a free cell

begin

initialize $I(w) \leftarrow 0$ for all $w \in W$

$I(v) \leftarrow 1$

for $i = 1$ **to** K

begin

Generate $D_i(v)$ which is the set of cells of distance i from v.

for every $w \in D_i(v)$

$$I(w) \leftarrow \max_{u \in D_{i-1}(v) \bigcap D_1(w)} I(v)$$

end

$$N(v) \leftarrow \sum_{w \in W} I(w)$$

end

end{**P1**}

The output of **P1**, N, is a matrix that contains the total number of cells visible from each cell v. According to **P1**, if the number of cells visible from any two cells next to each other is different, then the two cells belong to different regions. Another notable fact is that all the boundaries of a region are perpendicular to one or more principal axes. Therefore, the vertices, edges, and surfaces of regions can be determined from N as follows:

1. For any vertex, $\frac{\partial^3 N}{\partial x \partial y \partial z} \neq 0$.
2. For any edge parallel with z-axis, $\frac{\partial^2 N}{\partial x \partial y} \neq 0$. Similarly, edges parallel with x-axis or y-axis can be obtained.
3. For the surfaces that are perpendicular to x-axis, $\frac{\partial N}{\partial x} \neq 0$. Similarly, other surfaces can be determined.

P1 cannot detect the boundary between two adjacent regions when the total number of visible cells for the two regions happens to be the same (see Fig. 4.2). Such undetected boundaries can be easily recovered by extending some of detected boundaries. In Fig. 4.2, the boundary BC can be recovered later by extending either AB or CD.

By partitioning the workspace based on the relation $\sim_c$, the path planning problem can be divided into the following two subproblems.

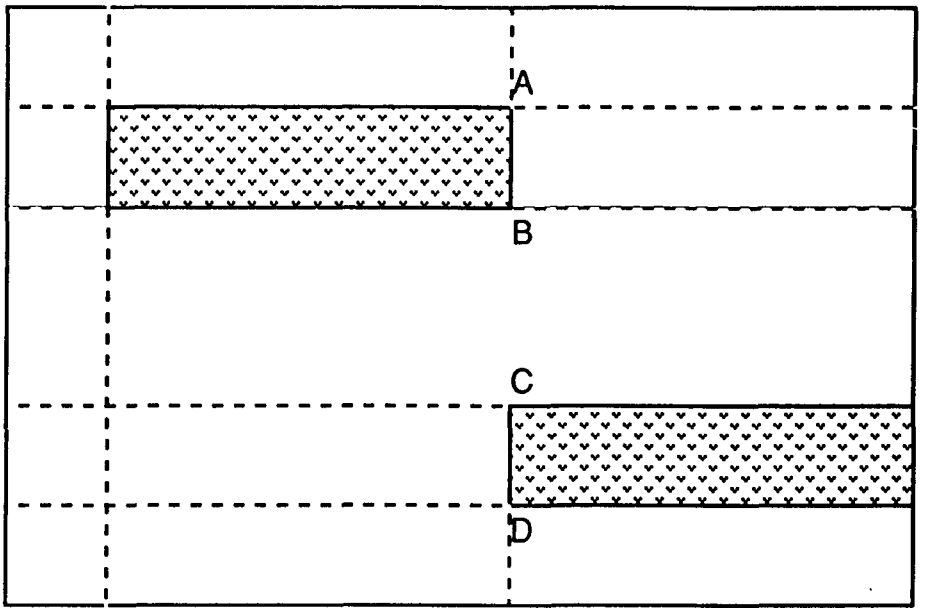


Figure 4.2: An example of undetected boundaries.

Subproblem 1 (Inter-Region) Find a sequence, $P = R_o R_1 R_2 \ldots R_k R_d$, of regions such that R_o is the region that contains the origin and R_d the region that contains the destination.

Subproblem 2 (Intra-Region) Find a path to traverse within each region in the sequence found from Subproblem 1.

4.3.1 Properties of Partitioned Regions

Before describing the properties of a partitioned region, it is necessary to define the following term.

Definition 7 *Search between two nodes is said to be* Free Of Backtracking *(FOB) if depth-first search can always find the shortest path between them without backtracking.*

If we know *a priori* the search between certain two nodes to be FOB, search efficiency can be improved greatly. However, it is very difficult, if not impossible, to

know this before the actual search takes place. The following lemma provides one useful instance of FOB.

Lemma 1 (Random-Path) For any two cells u and v such that $u >_c v$, the shortest path between u and v is FOB if the search started from v.

Proof: Let $D_k(v)$ be a set such that $D_k(v) = \{u : d_1(u,v) = k\}$, and w be a cell such that $w \in D_1(v) \cap D_{d_1(u,v)-1}(v)$. If no such w exists, v is not visible from u. This is impossible however, because v is always visible from itself, and thus, v should be visible from u by the definition of dominance. Thus, there always exists at least one cell, say w_1, such that $w_1 \in D_1(v) \cap D_{d_1(u,v)-1}(v)$. Since w_1 is visible from v, it is also visible from u by the definition of dominance. That is, there always exists a cell w_2 such that

$$w_2 \in D_2(v) \bigcap D_1(w_1) \bigcap D_{d_1(u,v)-2}(u).$$

Similarly, for any w_{i-1} there always exists $w_i \in D_1(w_{i-1})$ such that $w_i \in D_i(v) \bigcap D_{d_1(u,v)-i}(v)$ for $i = 2, 3, \ldots, d_1(u,v)$. ∎

Corollary 3 *For any u and v such that $u \sim_c v$, the shortest path between them is FOB regardless of the search direction used.*

According to Corollary 1, Subproblem **Intra-Region** can be solved trivially, i.e., the shortest path between two cells in the same region can be constructed by depth-first search. Furthermore, the shortest path between any two cells with dominance relation can be constructed by depth-first search without backtracking. In Fig. 4.3a, any cell in R_0 dominates all other cells in the workspace. Consider the construction of a path from a cell $p \in R_0$ to a cell $q \notin R_0$. Fig. 4.3b shows some of decision points during the search. Without knowing $p \sim_c q$, the search would start from p. The

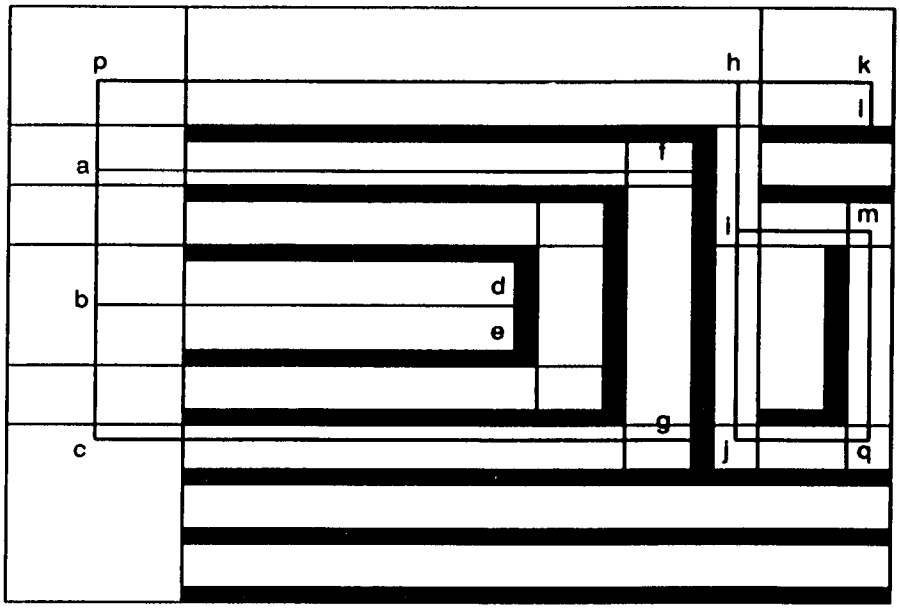
search may proceed towards a or h. If a is chosen, any subsequent search will end up with c or g and then fail. Even if h is chosen, the subsequent search may fail by choosing k instead of i as the next point. To remedy this problem, many algorithms are based on breadth-first search [59, 14] or best-first search [26, 15]. Hence, the computational complexity of these algorithms is $O(n^2)$ for 2D and $O(n^3)$ for 3D, where n is the number of decision points.

By contrast, if the search had started from q, there are still two directions to choose from: one towards j and the other towards m. However, the search proceeding towards m subsequently finds the shortest path between p and q; so does the search proceeding towards j. The absence of backtracking guarantees the success of depth-first search for a shortest path, thus resulting in computational complexity $O(n)$. It is necessary to have at least a 100×100 resolution to achieve acceptable accuracy. With the discretization resolution of 100×100 for 2D ($100 \times 100 \times 100$ for 3D), use of the dominance relation is shown to improve the search efficiency by a factor of 2 for 2D (4 for 3D) when the time taken to decide among several available directions is not considered.

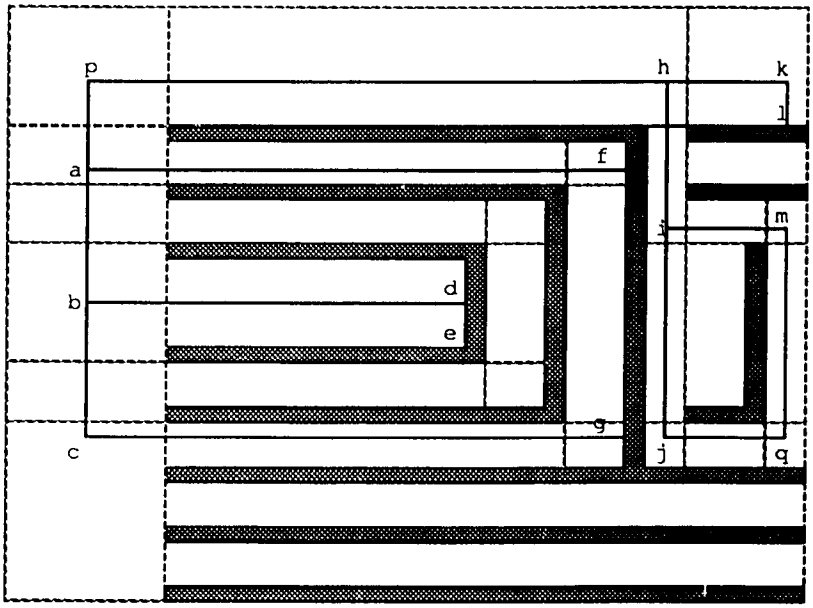
We now want to show how to determine the dominance relation between regions, and how to construct a path between regions with the dominance relation between them. To determine the dominance relation between regions, it is first necessary to understand the shape of each region.

Theorem 4 (2D case) Let v and w be two orthogonal neighbors[5] of a free cell u such that $v \sim_c w$. Then, $u \sim_c v$.

[5] An orthogonal neighbor of a node is the neighbor obtained as a result of applying an orthogonal operator to the node.



(a)



(b)

Figure 4.3: Dominance relation between the regions and its effect on the search.

Proof: Since $v, w \in D_1(u)$, there are four possible cases to consider: $v = n_x^+(u)$, $w = n_y^+(u)$; $v = n_x^+(u)$, $w = n_y^-(u)$; $v = n_x^-(u)$, $w = n_y^+(u)$; $v = n_x^-(u)$, $w = n_y^-(u)$.

Since one can prove the theorem similarly for all these four cases, it is sufficient to deal with one of them; we have chosen the first case. That is, we want to show both $v >_c u$ and $u >_c v$ when $v = n_x^+(u)$ and $w = n_y^+(u)$.

Consider the case of $v >_c u$ first. Let r be any cell visible from u, then there exists at least a sequence $uu_1u_2 \ldots u_{d_1(u,r)-1}r$ of free cells by the definition of visibility. Since $u_1 \in D_1(u)$, u_1 should be the neighbor of u in the positive/negative direction of x–axis or y–axis.

- When $u_1 = n_x^+(u)$, r is visible from v because $u_1 = v$.
- When $u_1 = n_y^+(u)$, r is visible from w because $u_1 = w$.
- When $u_1 = n_x^-(u)$, r is visible from v because there exists a sequence $vuu_1u_2 \ldots u_{d_1(u,r)-1}r$.
- When $u_1 = n_y^-(u)$, r is visible from w because there exists a sequence $wuu_1u_2 \ldots u_{d_1(u,r)-1}r$.

Therefore, any cell visible from u is also visible from either v or w; so is from v and w because $v \sim_c w$. Thus, $v >_c u$, i.e., $v, w >_c u$.

Now we want to show $u >_c v$. Let p be any cell visible from both v and w. Then, there exists at least a sequence $P_v = vv_1v_2 \ldots v_{d_1(v,p)-1}p$ of free cells. By the definition of visibility, there should be four possible generating operator sets, O, of P_v.

- When $O = \{n_x^+, n_y^+\}$, $uvv_1v_2 \ldots v_{d_1(v,p)-1}p$ will be the shortest path from u to p because $v = n_x^+(u)$. Hence, p is also visible from u.

- When $O = \{n_x^+, n_y^-\}$, $uvv_1v_2 \ldots v_{d_1(v,p)-1}p$ will be the shortest path from u to p because $v = n_x^+(u)$. Thus, p is also visible from u.

- When $O = \{n_x^-, n_y^+\}$, there exists at least one sequence $ww_1w_2 \ldots w_{d_1(w,p)-1}p$ of free cells between w and p because p is visible from w. Then, p can be reached from u via the sequence $uww_1w_2 \ldots w_{d_1(w,p)-1}p$ because $w = n_y^+(u)$.

- When $O = \{n_x^-, n_y^-\}$, if P_v consists of the cells obtained using the operator n_x^- then $v_1 = u$ (because there is a unique $n_x^-(v) = u$) else let v_k be the first cell such that $n_x^-(v_{k-1})$. Since v_{k-1} is visible from w, there exists a sequence $P_w = ww_1w_2 \ldots w_kv_k$ of free cells which can be obtained using the operator n_y^-. Since $n_y^-(w) = w_1 = u$, and thus, $uw_2 \ldots w_kv_kv_{k+1} \ldots v_{d_1(v,p)-1}p$ is a valid sequence for visibility, i.e., p is visible from u.

Therefore, $u >_c v$. ∎

Theorem 1 states an important fact that the shape of a region in 2D is always rectangular.

Corollary 4 (3D case) Let v, w, and x be three orthogonal neighbors of a free cell, u, such that $v \sim_c w$ and $v \sim_c x$. Then $u \sim_c v$.

Corollary 5 *In 2D space, there exists a rectangle that contains all the connected cells in the same region but no cells from other regions.*

Corollary 6 *In 3D space, there exists a rectangloid that contains all the cells in the same region and may also contain other embedded rectangloids.*

Corollary 3 provides valuable information on the whereabout of neighboring regions in 2D. It should be noted that neighboring regions of a region are always found

alongside its edges. Furthermore, the shortest path between two cells in neighboring regions passes through the projection of one of the two cells to an edge between the two regions. Any other path that does not pass through the projection will have the same or longer length. Note that there are four edges in a rectangle, and thus, there are at most four projections for each region as some of its edges may be occupied by obstacles. (See R_5 in Fig. 4.1.)

Unlike the 2D case, Corollary 4 implies a region in 3D to have an arbitrary shape (see Fig. 4.4.) This is due to the fact that one orthogonal neighbor of a cell may belong to a region different from the one that the other two [6] orthogonal neighbors belong to. Due to this irregular shape of region, it is very difficult to represent a region in 3D. One method of representing a 3D object is to enumerate its vertices, edges, and surfaces. This representation method is not attractive because of the difficulty in determining whether or not a cell belongs to a certain region. Moreover, enumerating all the members associated with a region could be costly due to the existence of a large number of cells in the region. The following lemma provides an important property of such an irregular region.

Lemma 2 *Let v_{ijk} and $v_{\ell mn}$ be any two cells such that $v_{ijk} \sim_c v_{\ell mn}$. For any cell v_{opq} such that $min(i,\ell) \leq o \leq max(i,\ell)$, $min(j,m) \leq p \leq max(j,m)$, $min(k,n) \leq q \leq max(k,n)$, $v_{opq} \Re v_{ijk}$ implies $v_{ijk} >_c v_{opq}$.*

Proof: For any cell u such that $u \Re v_{opq}$, there exists a path corresponding to the sequence of cells, $P_1 = uw_1w_2 \ldots w_{d_1(u,v_{opq})-1}v_{opq}$, generated by a set of orthogonal neighbor operators. Since $(\forall u\, u \Re v_{opq} \Rightarrow u \Re v_{ijk}) \Rightarrow v_{ijk} >_c v_{opq}$, let us suppose $u \not\Re v_{ijk}$. Let w_i be the cell such that $w_i \not\Re v_{ijk}$ and n be a neighbor operator such that $n(w_{i+1}) = w_i$ for some cell w_{i+1}. Since $v_{opq} \Re v_{ijk}$, there exists a generating set

[6] There are at most three orthogonal neighbors of a cell in 3D

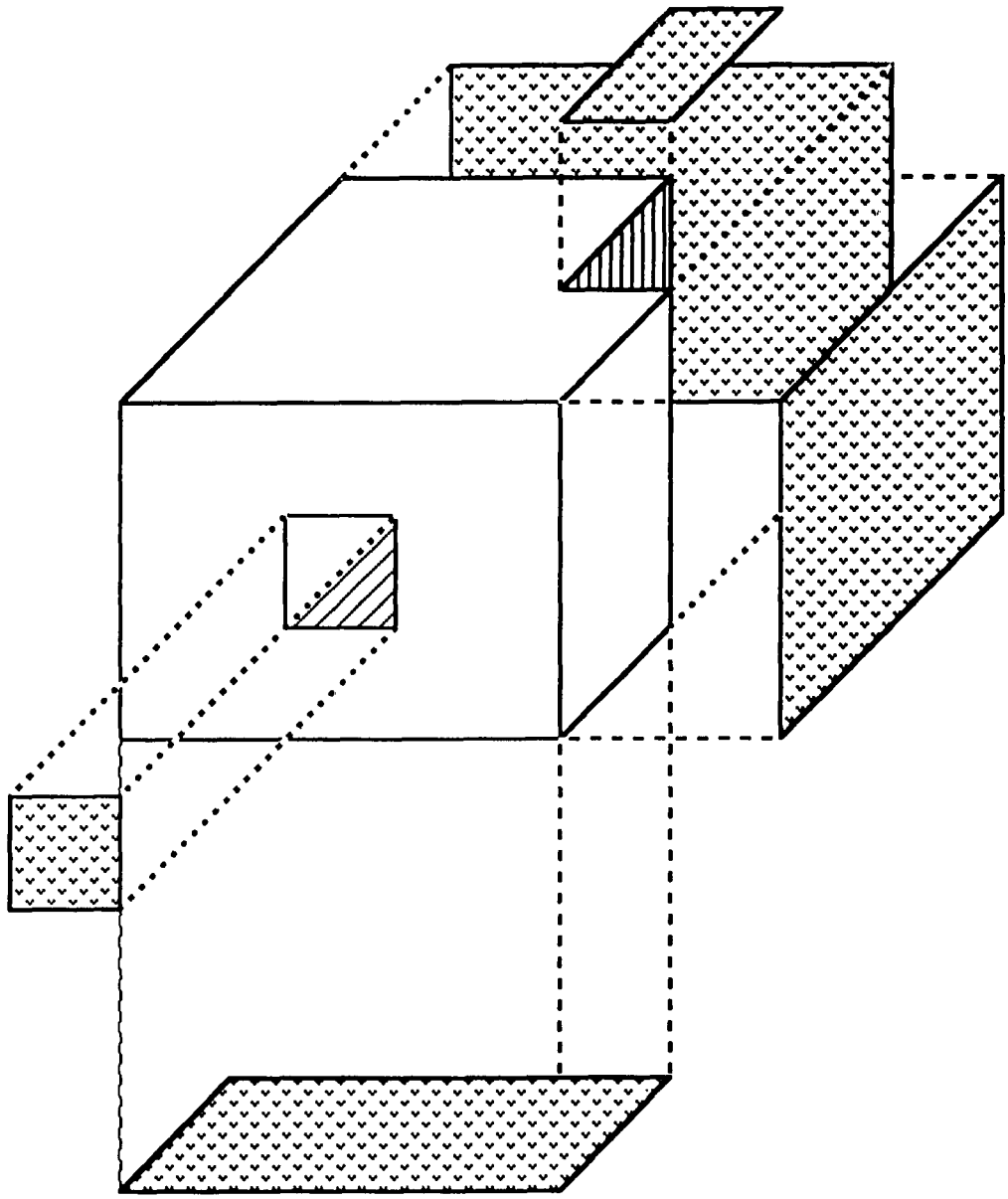


Figure 4.4: Typical shape of a region in 3D.

O_1 of operators for a shortest path P_1 from v_{opq} to v_{ijk}. Then, $\{n\} \cup O_1$ is not a set of orthogonal neighbors because $u \not\Re v_{ijk}$. Since $v_{ijk} \sim_c v_{\ell mn}$ and $v_{opq} \Re v_{ijk}$, there exists a generating set O_2 for a shortest path P_2 from v_{opq} to $v_{\ell mn}$. Then, $\{n\} \cup O_2$ should be an orthogonal set of neighbor operators because O_1 is the dual of O_2. This implies that $w_i \Re v_{\ell mn}$, contradicting the fact that $v_{ijk} \sim_c v_{\ell mn}$. Thus, $u \Re v_{ijk}$ and $v_{ijk} >_c v_{opq}$. ∎

The above lemma implies that when v_{opq} is not visible from v_{ijk} and $v_{\ell mn}$, it is completely isolated from the rectangloid formed by v_{ijk} and $v_{\ell mn}$. This is because all other cells within such a rectangloid are dominated by v_{ijk} and $v_{\ell mn}$, and thus, those cells not visible from v_{ijk} and $v_{\ell mn}$ are not visible from all other cells in the rectangloid either. In other words, any cell v that is visible from both v_{ijk} and v_{opq} is located outside the rectangloid. Therefore, the shortest path between v_{ijk} and v_{opq} should contain at least one cell outside the rectangloid.

Corollary 7 *For the smallest rectangloid containing a given cell u and for all the cells v such that $v \sim_c u$, $u \Re w \Rightarrow u >_c w$ for all cells w inside this rectangloid. Such a rectangloid is called the* Rectangloid of Dominance *(ROD) of u.*

Since the shape of region and/or ROD is a rectangloid, it is sufficient to represent the member cells in the region with the two extreme points $(x_{min}, y_{min}, z_{min})$ and $(x_{max}, y_{max}, z_{max})$. Whether a cell belongs to a region or not can easily be checked by comparing its location with these two points of the region or ROD. These two points will henceforth be called the *range* of region R and denoted by $r_{min}(R)$ and $r_{max}(R)$. Using the range, the *cover* relation is defined as follows.

Definition 8 *For any two regions R_1 and R_2, R_1 is said to* cover *R_2, denoted by $R_1 \rhd R_2$, when*

$$r_{min}(R_1) \leq r_{min}(R_2),\ r_{max}(R_2) \leq r_{max}(R_1), R_1 \neq R_2.$$

4.3.2 Workspace Representation

Dominance relations among regions can be represented as a graph and so can the workspace.

Definition 9 *The workspace is represented as a digraph, $G = (V, E)$, where V is the set of regions and E is the set of edges such that $\exists$ an edge e from $R_1 \in V$ to $R_2 \in V$ if and only if $R_1 \rhd R_2$.*

There are two sources of difficulty to obtain the dominance graph (DG): (i) it is difficult to check the dominance relation between all pairs of regions due to the large number of possible combinations, and (ii) it is difficult to describe a 3D region due to its irregular shape.

To circumvent these difficulties, a modified dominance graph (MDG) is defined as follows.

Definition 10 *The MDG is a digraph, $MDG = (V, E')$, where V is the set of regions and E' is the set of edges such that $\exists$ an edge e from $R_1 \in V$ to $R_2 \in V$ if and only if $R_1 \rhd R_2$, $R_1 \neq R_2$, and there is no $R \in V$ such that $R_1 \rhd R$ and $R \rhd R_2$.*

Notice that a MDG contains partial information on the dominance relation for a given workspace. Especially, $E' = \emptyset$ for 2D as shown in Corollary 3. A similar example can also be found in Fig. 4.3. Though R_0 dominates all other regions in the workspace, it will not be shown in the MDG. However, this will not cause any problem

since the main purpose of MDG is to find a shortest path in 3D. Fig. 4.5 shows an example workspace with two obstacles and the same workspace after partition. Then, the workspace is converted into DG and MDG as shown as in Fig. 4.6. Notice that most of the dominance relations in DG are shown in MDG except those of two regions $\{(0,0,0),(18,15,2)\}$ and $\{(19,0,0),(20,15,2)\}$. This is due to the limited range of those two regions.

We have shown that the shortest path can be found with depth-first search when a dominance relation exists between the origin and destination. In some cases, however, no dominance relation may exist between a given pair of origin and destination. Consider the problem of finding a shortest path between two cells u and v such that $u \not\succ_c v$ and $v \not\succ_c u$. Let $R(u)$ be a region containing the cell u. The shortest path between u and v should contain at least a cell from one of the regions next to $R(u)$. Such regions will henceforth be called *bordering regions* of $R(u)$. The closest cell that belongs to a bordering region of $R(u)$ can be found by projecting u to its borders. There exist at most 4 projections in 2D and 6 projections in 3D because the shape of region (and ROD) is rectangular (see Fig. 4.7.). Suppose the shortest path $P(u,v)$ between u and v passes through $R(w)$, one of $R(u)$'s bordering regions where w is the projection of u. Then one of the following cases is true.

1. w is visible from u.

2. w is not visible from u.

3. w is occupied by an obstacle.

In Case 1, $P(u,v)$ can be obtained by concatenating $P(u,w)$ and $P(w,v)$ in L_1-metric. Note that $P(u,w)$ is a straight-line segment between u and w; otherwise, w

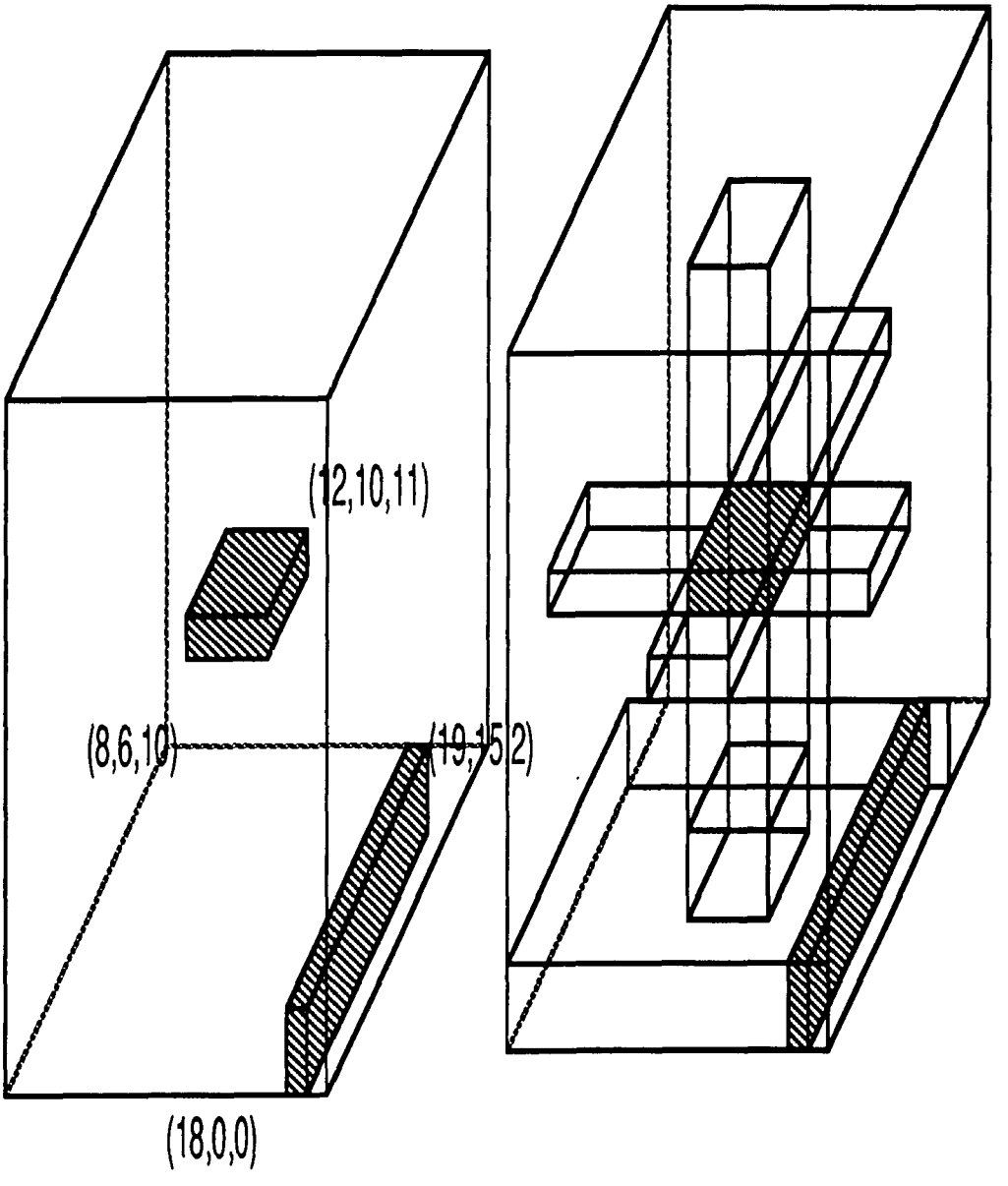
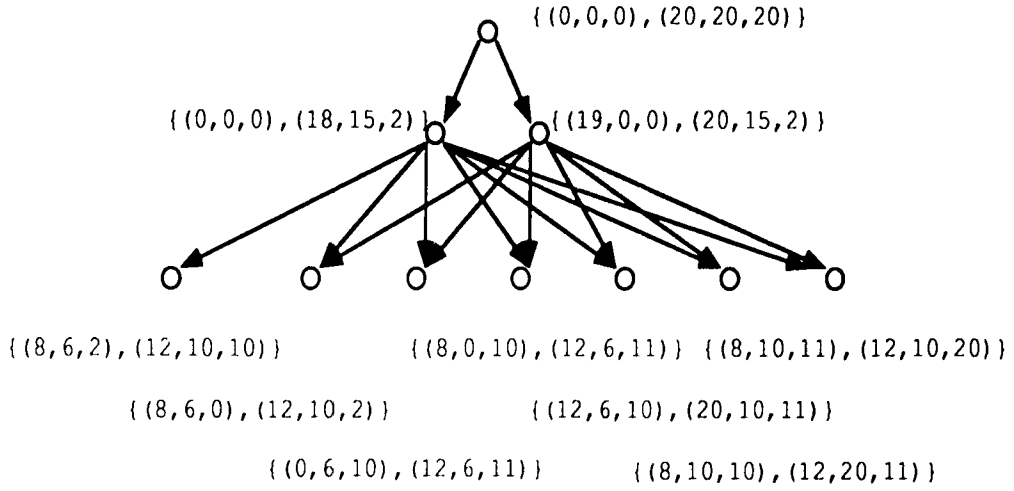


Figure 4.5: Generation of 3D regions.

DG



MDG

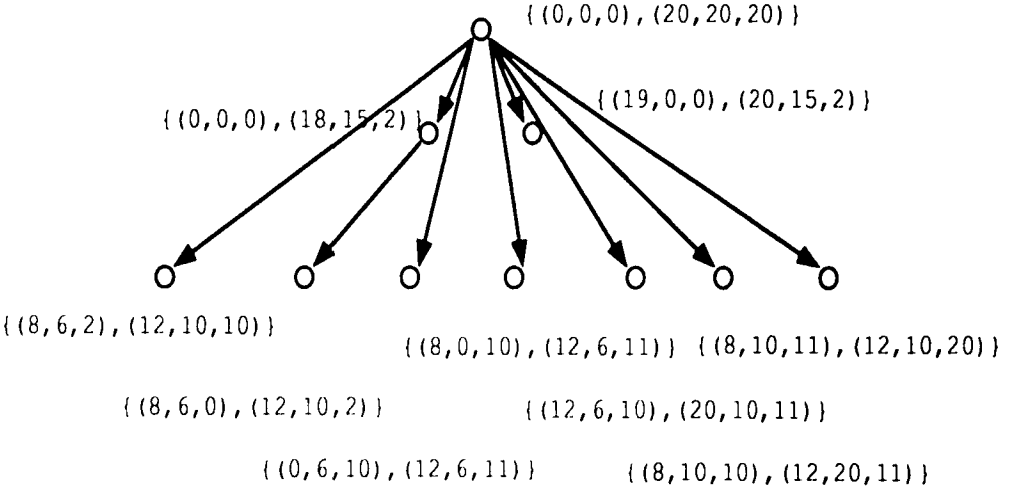


Figure 4.6: DG and MDG.

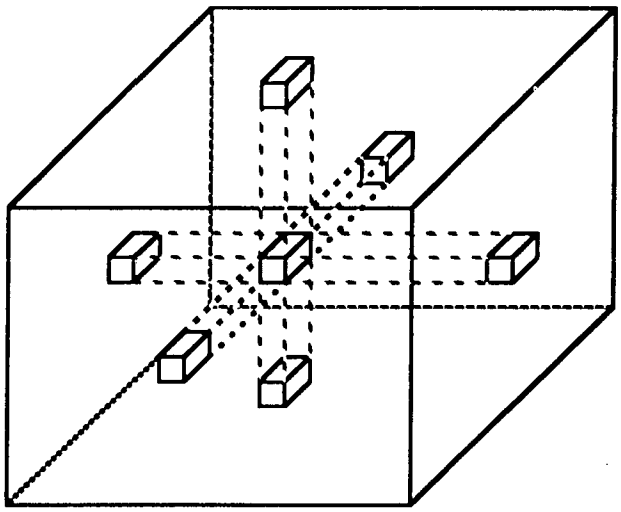


Figure 4.7: Projections of a search point.

cannot be visible from u. Case 2 cannot be true since no cell in $R(w)$ can be visible from u and $R(w)$ is not next to $R(u)$.

Fig. 4.8 shows an example of Case 3. Among 4 projections a, b, c, and d of a cell u, b and c are occupied by an obstacle. Unlike b, we may have to find a replacement cell $x \in R(c)$ for c that is closest to u but not inside the obstacle. Finding such a cell may be difficult as, in many cases, such a cell is not unique in 3D. It should be noted that we need a replacement for c only when shortest path should pass through the border of $R(u)$ and $R(c)$. For example, it is not necessary to find a replacement for c when the destination is e as the shortest path can pass d. On the contrary, the shortest path between u and f should pass through the common border of $R(u)$ and $R(c)$. That is the case when $R(u)$, $R(c)$, and $R(f)$ are separated by obstacles that are located completely outside $R(u) \cup R(c) \cup R(f)$. Such obstacles do not interfere with the path between u and f and can thus be ignored. In other words, construction of $P(u, f)$ is FOB when starting from u.

The following algorithm constructs a shortest path between u and v for the general case. Informally, after initialization, the algorithm examines the MDG to see whether

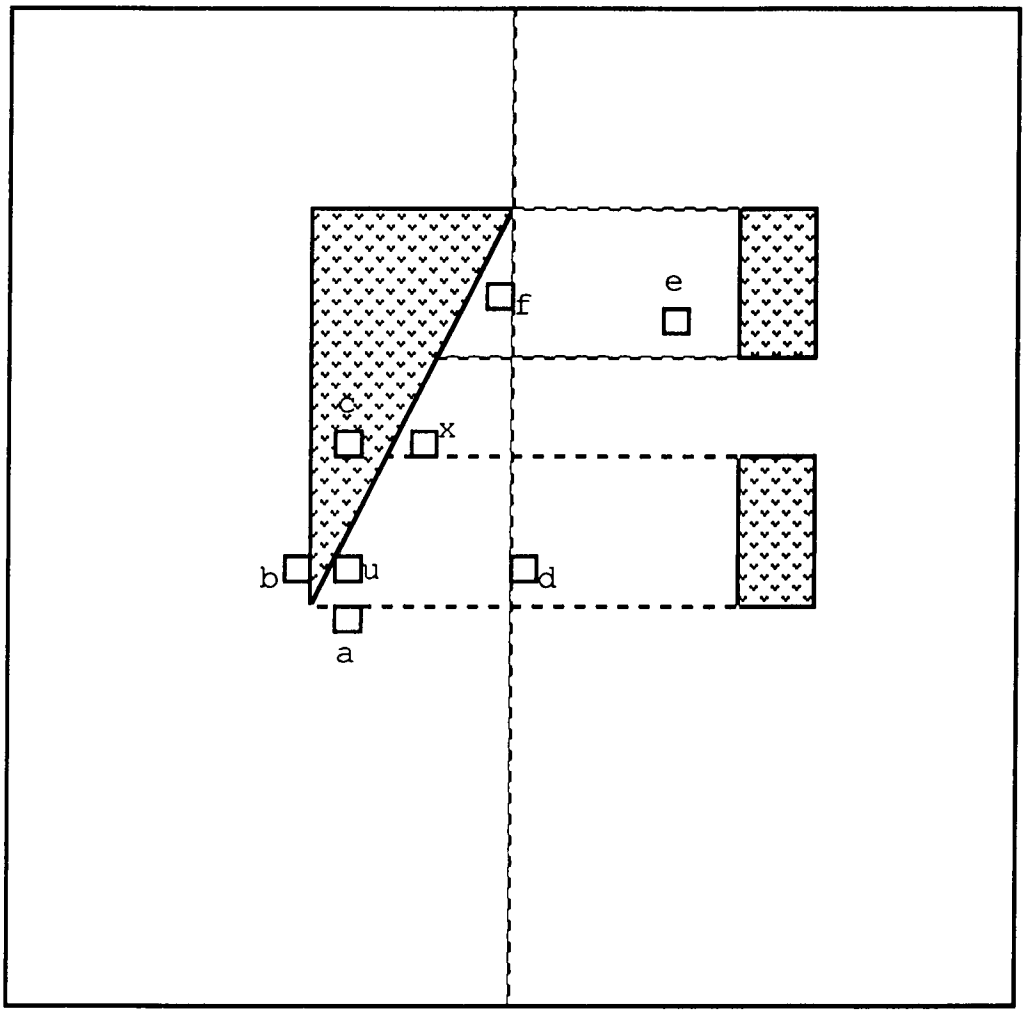


Figure 4.8: Regions separated by an obstacle outside their RODs.

there exists any dominance relation between the current cell (initially, the origin) and the destination. If there is, the algorithm constructs the path using depth-first search. Otherwise, a set T of projections of the current cell is obtained. For each member of T, check whether it is occupied by obstacle or not. If it is occupied, check whether construction of a path between the destination and the current cell is FOB or not. If so, the algorithm stops after constructing the path. Otherwise, that projection is deleted from T, the remaining members of T are added to S, the set of cells yet to be examined, and the best cell is added to U, the set of examined cells. Then, we choose the most attractive cell (the closest cell to the destination) in S as the current cell and the procedure repeats itself until path construction is completed or S becomes empty. What we said above can be summarized in algorithm form as follows.

1. Let $Best := u$, $P(u, Best) =$ nil, $S := \emptyset$, $T := \emptyset$ and $U := \emptyset$.

2. If $Best >_c v$ then construct $P(Best, v)$ and go to Step 8.

3. If $v >_c Best$ then construct $P(Best, v)$ and go to Step 8.

4. $T :=$ { Projections of $Best$ } $- S$. For every $w \in T$,

   If w is occupied by an obstacle then try to construct $P(Best, v)$

   using depth-first search.

   If path construction is successful then go to Step 8. Else $T := T - \{w\}$.

   else $P(u, w) :=$ concat$(P(u, |current_cell|), P(Best, w))$.

5. $S := S \cup T$, $S := S - \{Best\}$ and $U := U \cup \{Best\}$.

6. Let $Best$ be such that $\min_{Best \in S}(\text{length}(P(u, Best)) + d_1(Best, v))$.

7. Go to Step 2.

8. $P(u, v) := \text{concat}(P(u, Best), P(Best, v))$.

The computational complexity from Step 2 to Step 4 is $O(n)$ where n is the resolution of the workspace, i.e., the number of cells in each axis. As the algorithm stops when either path is found or S is empty, the maximum number of iterations from Step 2 to Step 5 occurs when S is empty. That is, the maximum number of iterations is identical to the total number of regions, m, and the overall complexity becomes $O(mn)$. Since the total number of regions can be as high as the total number of cells (i.e., $m = O(n^3)$), the overall complexity can be as high as $O(n^4)$. This overall complexity is deceiving as the total number of regions is much smaller than the total number of cells (i.e., $m << n^3$). Since RODs, rather than individual regions, are searched, search efficiency is also improved.

4.4 An Example Workspace

In this section, we consider an example workspace cluttered with various shapes of obstacles as shown in Fig. 4.9. Specifically, the effects of various orientations of an obstacle on workspace partitioning are described and a typical path in such an environment is also constructed.

The workspace is digitized as a $32 \times 32 \times 32$ grid. The resulting MDG has statistics as shown in Table 4.1. Region sizes vary from one to several thousand cells. Most one-cell regions are due to the pyramid shape obstacle K_2. Diagonal edges in K_2 usually divide the workspace into small regions. However, diagonal edges in K_4 do not contribute to the fragmentation of the workspace at all. Due to these one-cell regions, the median region size is 2 while the mean region size is 71.6.

According to our simulation based on 1,000 randomly selected origin-destination pairs, the average number of regions searched is less than three. For 63.6% of the

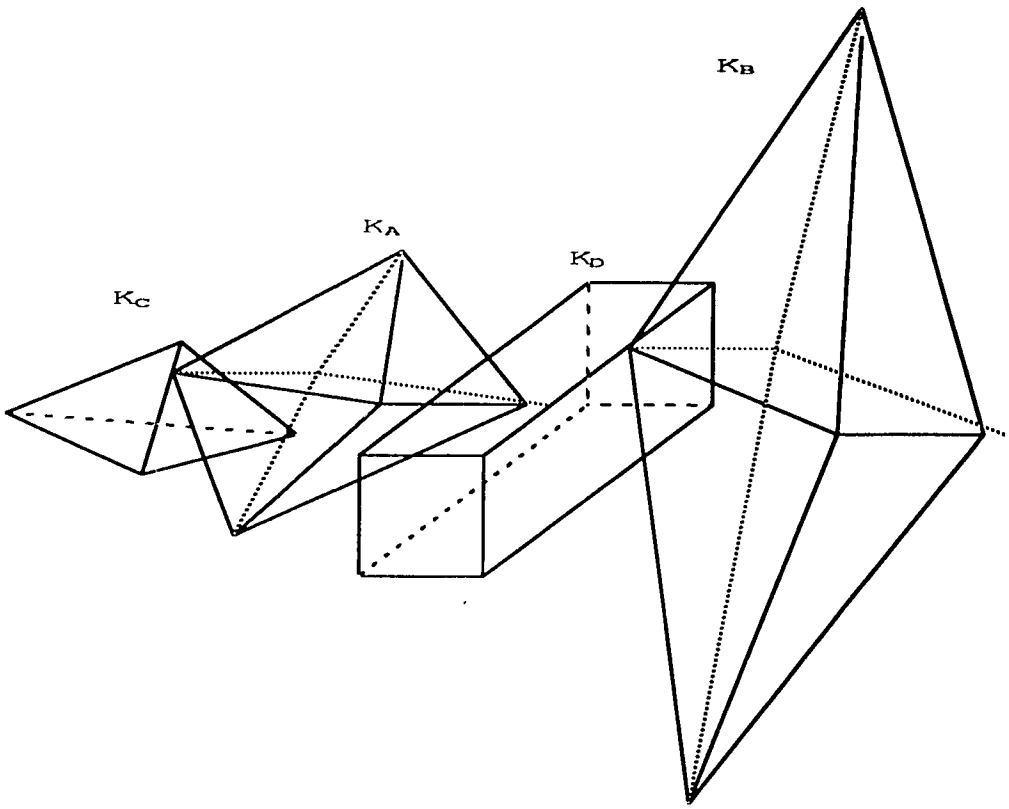


Figure 4.9: A sample workspace with various obstacles.

cases considered, there exists a dominance relation between the region containing the origin and that containing the destination, thus requiring no region to be searched at all. This is mainly due to the fact that the largest region dominates all the other regions and requires no regions to be examined, and on the average, either the origin or the destination lies in the largest region for approximately 50% of the time. Furthermore, only one region needs to be searched for 28.2% of the time. This implies that we need to search less than two regions for 91.8% of the time.

The fragmentation has little effect on path construction, i.e., the probability of the destination or origin falling in a one-cell region is less than 0.01. Even in the case when the origin falls in a one-cell region, the number of regions to be searched is very small if the destination belongs to a larger region by changing the search direction. For the source-destination pair (A, B) in Fig. 4.9, the total number of regions examined is zero since there exists a dominance relation between A and B even though B falls in a one-cell region. The worst-case occurs when both the origin and destination fall in one-cell regions and are placed on opposite side of K_2, shown as C and D in Fig. 4.9, respectively. In such a case, the total number of regions searched can be as high as 200. However, the probability of such a case occurring is less than 0.0001.

4.5 Summary

In this chapter, we presented a new method of partitioning the workspace using L_1-visibility. It was shown that the optimal path with respect to L_1-metric between two partitioned regions can be obtained easily if a dominance relation exists between them. When no such relation exists between the origin and the destination, we have presented an $O(mn)$ algorithm. Our path planner is shown to find an optimal solution

Total number of cells	32,768
Total number of cells occupied by obstacles	3,550
Total number of free cells	29,218
Total number of regions	408
Size of the largest region	8,857
Size of the smallest region	1
Median region size	2
Mean region size	71.6
Avg. number of regions examined	2.23

Table 4.1: Statistics of regions.

for the digitized workspace. Though the workspace with polyhedral obstacles are regarded as a more general solution, many workspace configurations are obtained in digitized form and our algorithm provides a very efficient solution in such an environment.

This chapter focussed on the shortest path planning in 3D. Unlike other approaches, our method does not depend on any particular geometry. Since each region is represented with two extreme points or inequality predicates, our algorithm can be extended to k Dimensional space for $k \geq 4$ without much difficulty.

CHAPTER V

A PROBABILITY FIELD APPROACH TO ROBOT PATH PLANNING

5.1 Introduction

One of the major factors to be considered in automatic path planning is obstacle information. In many circumstances, it is not easy to obtain accurate obstacle information. This may be due to the existence of moving obstacle [8], or a huge-size environment [86]. The latter is usually the case of mobile robot path planning [26] where the environment is usually huge compared to the size of the robot and often includes other mobile robots. In such environments, it is not wise to use algorithms based on the structure of obstacles such as [25, 3] because these algorithms require precise knowledge of the obstacles in the workspace.

Some of the most popular algorithms in such environments are based on the potential field approach [31, 20]. The potential field approach is based on the map generated by the superposition of imaginary forces: repulsive forces generated from obstacles and attractive forces generated from goals/destinations. Superposition of imaginary forces make it possible to dynamically generate each force. Another advantage of the potential field approach is the use of field data to directly control actions during navigation. It is shown, however, that this approach has the drawback

of getting into local minima [35]. To remedy this drawback, Borenstein and Koren [4] proposed vector-force field algorithm by integrating two known concepts: certainty grids for obstacle representation [56], and potential fields for navigation. However, it does not solve the problem completely because a certainty grid is the representation of inaccurate sensory data about obstacles.

A gradient-field approach proposed by Payton [63, 64] deals with this problem by calculating the cost of each grid cell of a digital map. The cost of each cell is based on the score obtained by applying a search algorithm such as A^* [19], or Dijkstra's algorithm [15]. Since the gradient-field approach is useful only in a known environment, it cannot deal with changing environments efficiently due to the excessive computational requirements. Zhao [85] proposed an algorithm that can deal with both known and unknown environments using a heuristic-search method (recovery algorithm) based on the $A*$ algorithm. It was shown that the efficiency of Zhao's algorithm largely depends on the scale factor of the map.

When a local minimum is encountered during the search, there are basically two solutions depending on the search strategy used:

(i) Path runs sideways or backward trying to find the position closer to the destination (Hill-Climbing).

(ii) Backtrack to a position which was generated before and neighbors at least one unexplored free position that is closer to the destination (Best-First).

In each of these cases, the search will either increase the path length or waste a certain number of steps.

Our main strategy is to minimize the number of times a search has to backtrack, i.e., the search meets a local minima, or a deadend. For this purpose, we defined two events 1) a digitized cell is a local minima and 2) a digitized cell may lead to a

local minima in subsequent moves. By formulating the probability of these events, we form a map of probability field and present an algorithm utilizing the field data. In contrast to the previous gradient-field approaches, the probability field is based on the recursive definition of the event when a cell leads to a deadend. It is shown that our algorithm converges very quickly and very computationally efficient.

This chapter is organized as follows. Section 5.2 describes the terminology used along with a formal definition of the deadend. Section 5.3 describes the definition of a probability field and necessary formula to compute the probability fields. We also present there an algorithm that computes the probability field. In Section 5.4, we consider two sample workspaces and analyze the effects of the probability field as compared to other information such as the distance from obstacles using several search strategies. This chapter concludes with Section 5.5.

5.2 Terminology

For a given destination, a position is said to be a *deadend* when (i) all of its neighbors are occupied by obstacles, or (ii) moving to any of its unoccupied neighbors increases the path length. A position is said to *meet a deadend* when it is a deadend or its chosen neighbor meets a deadend.

Since our algorithm is not limited to 2D workspace, the notations are based on a 3D workspace. We assume that a 3D workspace consists of $l \times m \times n$ identical cubes. In addition, we will use the following symbols throughout this chapter.

W : The set of all cubes in the workspace.

W_{obs} : The set of all cubes occupied (completely as well as partially) by obstacles.

v_{ijk} : The (i,j,k)-th cube in the workspace.

$D_l(v)$: The set of all cubes whose L_1 distance is l from a cube v.

$\hat{V}_{ijk}$: The set of destinations for which v_{ijk} becomes a deadend. That is, $\hat{V}_{ijk} = \{w : u \in D_1(v_{ijk}), d_1(u,w) = d_1(v_{ijk},w) - 1 \Rightarrow u \in W_{obs}\}$.

B_{ijk} : The event that v_{ijk} becomes a deadend.

C_{ijk} : The event that the search meets a deadend if a cube v_{ijk} is chosen.

R_{ijk}^{lmn} : The event that v_{ijk} precedes v_{lmn} on a path. These two cubes are not necessarily neighbors to each other on the path.

g_{ijk}: The probability that v_{ijk} becomes a destination.

p_{ijk} : The probability that v_{ijk} becomes a deadend.

q_{ijk} : The probability that the search will meet a deadend later if the path passes through v_{ijk}.

5.3 Definition of Probability Field

Obviously, only those cubes that are adjacent to obstacles could be deadends. The average probability that a point v_{ijk} becomes a deadend is calculated by:

$$p_{ijk} = \sum_{v_{lmn} \in \hat{V}_{ijk}} g_{lmn}.$$

For example, if destinations are uniformly distributed throughout the workspace, then

$$g_{ijk} = \frac{1}{|W| - |W_{obs}|}$$

for all destinations in $\hat{V}_{ijk}$, where $|W|$ is the cardinality of the set W. Then, the probability of v_{ijk} becoming a deadend is

$$p_{ijk} = \frac{|\hat{V}_{ijk}|}{|W| - |W_{obs}|}.$$

The probability that a path containing v_{ijk} meets a deadend can be calculated by:

$$P[C_{ijk}] = P[\bigcup_{v_{lmn} \in W} (R_{ijk}^{lmn} \bigcap B_{lmn})].$$

Since a path consists of a sequence of successive neighboring points, we get

$$P[C_{ijk}] = P[B_{ijk} \bigcup_{v_{lmn} \in W} \{(\bigcup_{v_{opq} \in D_1(v_{ijk})} R_{ijk}^{opq} \bigcap R_{opq}^{lmn}) \bigcap B_{lmn}\}].$$

The event that a point becomes a deadend and the event that a point leads subsequently to a deadend are mutually exclusive. Thus, the above equation becomes:

$$\begin{aligned}
P[C_{ijk}] &= P[B_{ijk}] + P[\bigcup_{v_{lmn} \in V} \{(\bigcup_{v_{opq} \in D_1(v_{ijk})} R_{ijk}^{opq} \bigcap R_{opq}^{lmn}) \bigcap B_{lmn}\}] \\
&= p_{ijk} + P[\bigcup_{v_{lmn} \in V} \{(\bigcup_{v_{opq} \in N(v_{ijk})} R_{ijk}^{opq} \bigcap R_{opq}^{lmn}) \bigcap B_{lmn}\}] \\
&= p_{ijk} + P[\bigcup_{v_{opq} \in N(v_{ijk})} \{(\bigcup_{v_{lmn} \in V} R_{ijk}^{opq} \bigcap R_{opq}^{lmn}) \bigcap B_{lmn}\}] \\
&= p_{ijk} + P[\bigcup_{v_{opq} \in N(v_{ijk})} \{R_{ijk}^{opq} \bigcup (\bigcup_{v_{lmn} \in V} (R_{opq}^{lmn} \bigcap B_{lmn}))\}] \\
&= p_{ijk} + P[\bigcup_{v_{opq} \in N(v_{ijk})} (R_{ijk}^{opq} \bigcap C_{opq})]. \qquad (5.1)
\end{aligned}$$

Since the search algorithm always chooses only one neighbor at a time, the events R_{ijk}^{opq}'s for all $v_{opq} \in N(v_{ijk})$ are mutually exclusive. Thus, Eq. (5.1) becomes:

$$\begin{aligned}
P[C_{ijk}] &= p_{ijk} + \textstyle\sum_{v_{opq} \in N(v_{ijk})} P[R_{ijk}^{opq} \bigcap C_{opq}] \\
&= p_{ijk} + \textstyle\sum_{v_{opq} \in N(v_{ijk})} P[R_{ijk}^{opq} | C_{opq}] P[C_{opq}]. \qquad (5.2)
\end{aligned}$$

It should be noted that R_{ijk}^{opq} depends on the search strategy used. For example, in a blind search, R_{ijk}^{opq} is independent of C_{opq}. When R_{ijk}^{opq} is independent of C_{opq}, R_{ijk}^{opq} can be obtained by the same method used to calculate p_{ijk}.

The most logical search strategy that utilizes the probabilities of meeting deadends must choose a point least likely to lead to a deadend as long as such a choice does not increase the path length. However, it is difficult to derive $P[R_{ijk}^{opq} | C_{opq}]$ in

Eq. (5.2) because C_{opq} is unknown and depends on R_{ijk}^{opq}. Moreover, the path cost depends on the probabilities of meeting deadends. To overcome these difficulties, it is necessary to introduce a new event, denoted by C_{ijk}^{n}, that v_{ijk} will meet a deadend in n steps. Then, Eq. (5.2) can be rewritten as:

$$P[C_{ijk}^{n+1}] = \begin{cases} p_{ijk} & \text{if n = 1} \\ p_{ijk} + \sum_{v_{opq} \in N(v_{ijk})} P[R_{ijk}^{opq} | C_{opq}^{n}] P[C_{opq}^{n}] & \text{if } n \geq 2. \end{cases} \tag{5.3}$$

By using mathematical induction and Eq. (5.3), $P[D_{ijk}^{n}]$ can be calculated for any $n \geq 2$. Since $D_{ijk} = \lim_{n \to \infty} D_{ijk}^{n}$, we can derive $P[D_{ijk}]$ by applying Eq. (5.3) recursively.

The path planner consists of two phases. In the first phase, the workspace information is transformed into the probabilities of meeting deadends which are computed off-line as discussed above. 2D examples of the output of the first phase using the number of points in a path as the path cost are shown in Fig. 5.1 and Fig. 5.3. In these examples, the workspace consists of 32×32 points and the neighbor of a point in the workspace is defined as the set of all points that are horizontally, vertically, and diagonally adjacent to the point. Since robot motions are usually performed through rotary joints, diagonal movements are more natural than strict horizontal and/or vertical movements in many cases. The destinations used in these examples are assumed to be distributed uniformly[1] over the entire workspace. 108 points in Fig. 5.1 are occupied by the obstacles while 144 points in Fig. 5.3 are occupied by the obstacles, i.e., more obstacles in Fig. 5.3 than in Fig. 5.1. The obstacle data is shown on the left and $P[D_{ijk}]$ at the right of these figures.

Upon receiving the workspace information, the first phase of the path planner is to transform the obstacle data into the probabilities of meeting deadends using the

[1] Any other distributions can be assumed.

procedure **Calc_Prob** below.

Procedure **Calc_Prob**

1. Compute p_{ijk} for all $v_{ijk} \in W$ using $p_{ijk} = \frac{|\hat{V}_{ijk}|}{|W| - |W_{obs}|}$.

2. Initialize $P^*[C_{ijk}] := p_{ijk}$, $error := 0.001$, and $max\_error := \infty$.

3. Repeat 3a-c until $max\_error < error$.
   3a. For all $v_{ijk} \in V$, compute $P^*[R_{ijk}^{opq} \mid C_{opq}]$ using $P^*[C_{opq}]$, and $P[C_{ijk}] := p_{ijk} + \sum_{v_{opq} \in N(v_{ijk})} P^*[R_{ijk}^{opq} \mid C_{opq}] P^*[C_{opq}]$.
   3b. Set $max\_error := \min(max\_error, \max_{v_{ijk} \in V} (\mid P(C_{ijk}) - P^*[C_{ijk}] \mid)$.
   3c. Set $P^*[C_{ijk}] := P[C_{ijk}]$.

The second phase of the path planner is to convert each origin-destination pair to a collision-free optimal path. The hill-climbing method is chosen as our search strategy. The main advantage of the hill-climbing method is that it may converge to a good solution very quickly. However, it may waste a great deal of the time when a successor that leads to a local maximum or a deadend is chosen. Since our algorithm is designed to choose a successor that is least likely to lead to a deadend, this disadvantage of the hill climbing method is minimized.

Although our method chooses a point that will least likely lead to a deadend, it may still lead to a deadend. Furthermore, there are some origin-destination pairs that will always lead to deadends due to, for example, a large obstacle between them. One way of dealing with this problem is to backtrack to an earlier point and choose some other directions from there on. Decisions to be made in the backtracking are: the length of backtrack (measured in the number of points/cubes to backtrack) and the existence of a path without going sideways. The former is important because a smaller number of backtracking steps will often lead to the same deadend, while a larger number of backtracking steps will waste the search time and may also lead to another deadend that the original path has already avoided. The latter presents a

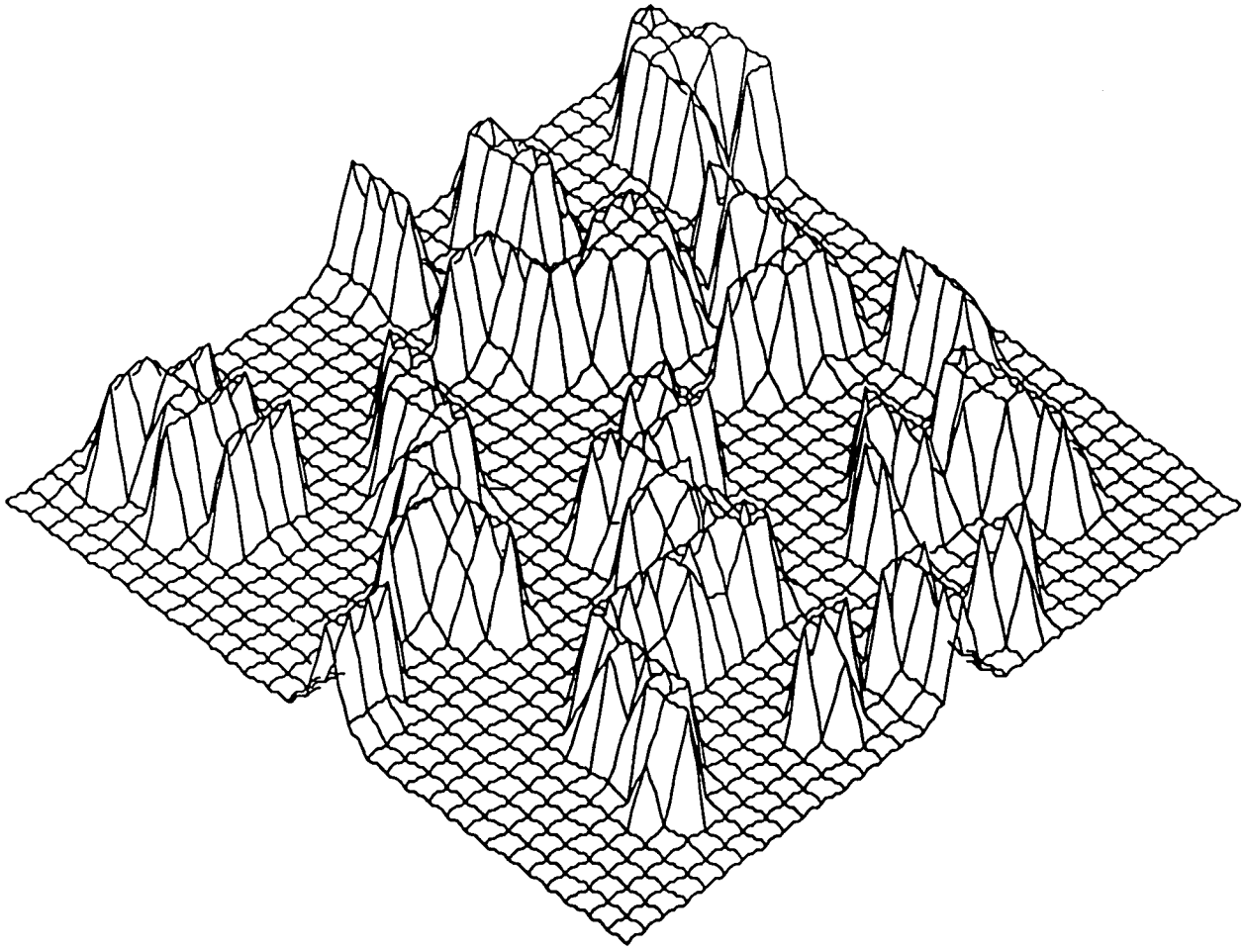


Figure 5.1: Example 1 of workspace being transformed into probability fields.

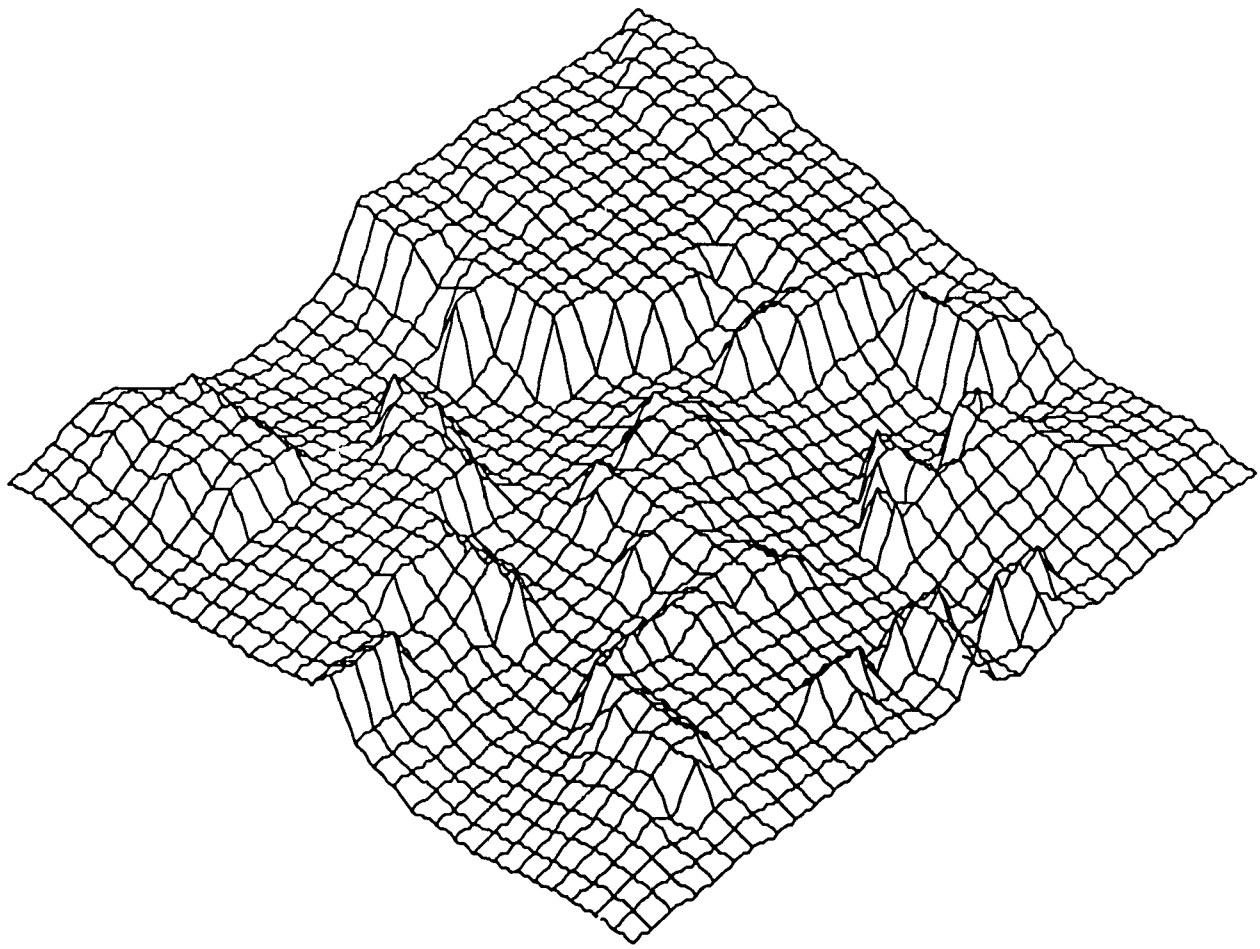


Figure 5.2: The probability field data obtained from example 1.

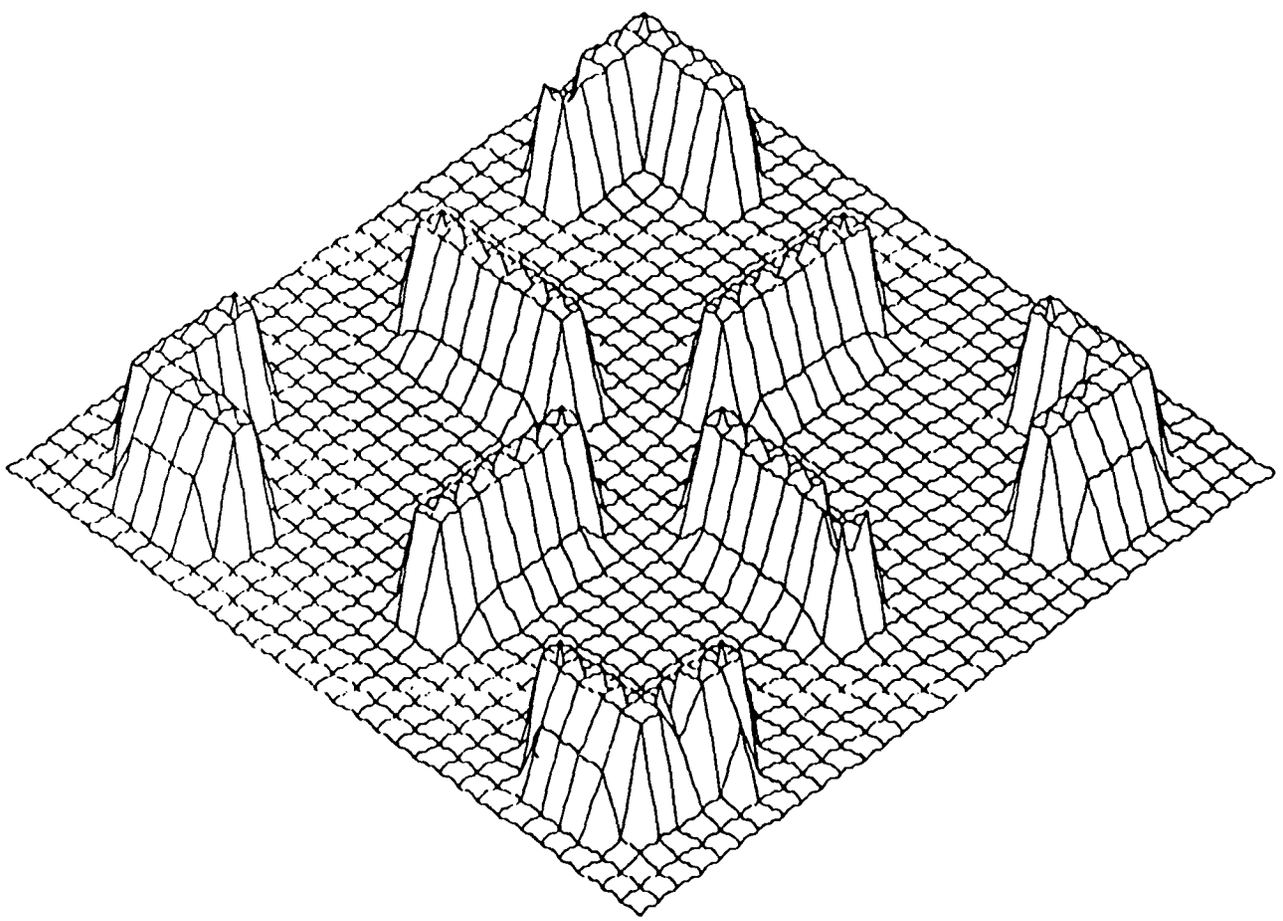


Figure 5.3: Example 2 of workspace being transformed into probability fields.

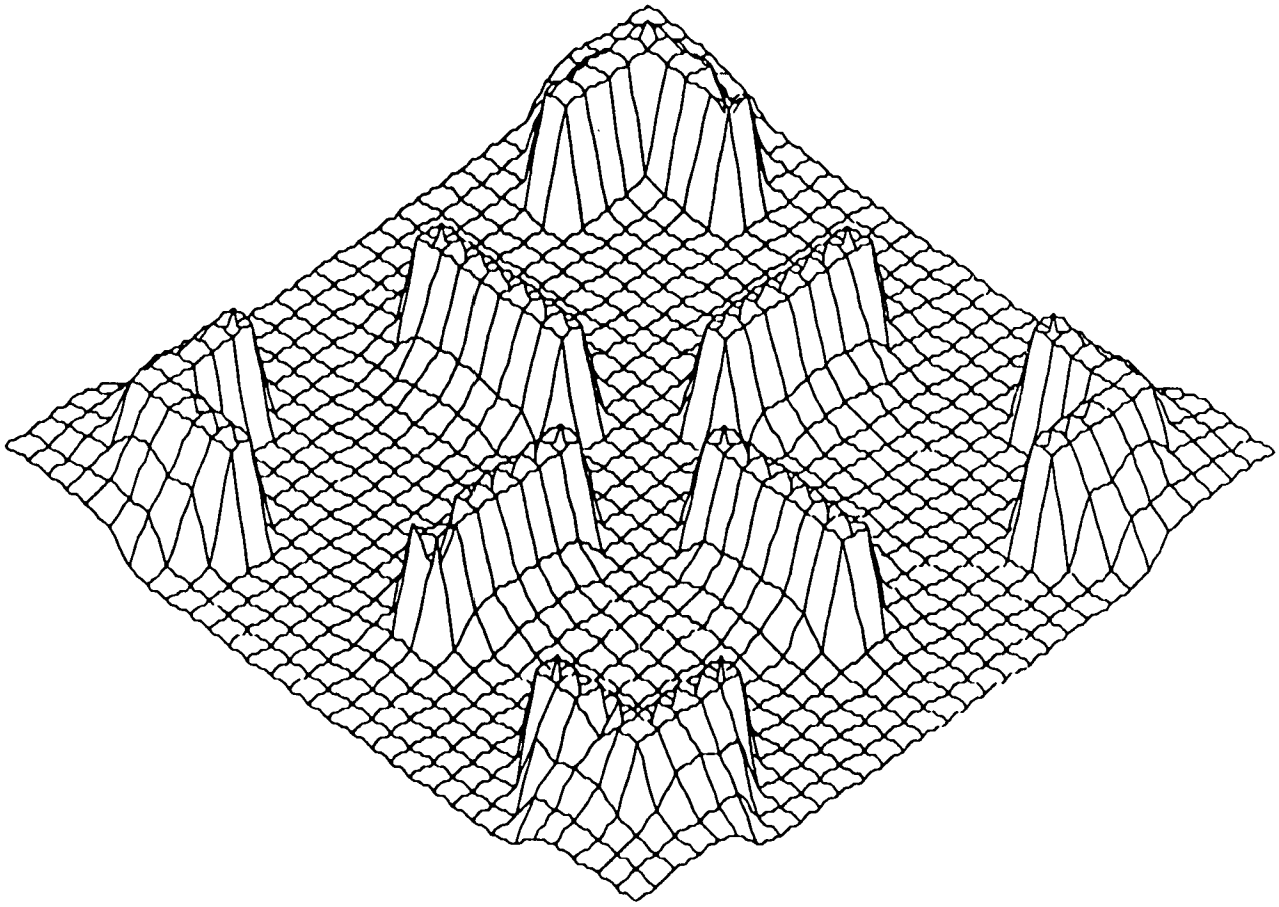


Figure 5.4: The probability field data obtained from example 2.

more critical problem than the former because the existence of a path can be verified only after testing all combinations of points. According to our simulation results, our method is shown to avoid deadends very well when there exists a path connecting the origin and the destination without moving sideways. That is, if the first attempt to find a path for a given origin–destination pair meets a deadend, the second attempt usually ends up meeting a deadend again.

To remedy the above situation, we adopted the following backtracking policy. For a partially constructed path $P_k = v_0 v_1 \cdots v_k$, where v_0 is the origin, v_k the point that becomes a deadend, and let q_i be the probability of v_i leading to a deadend.[2]

Step 1 Find the first v_i' such that v_i' is a sibling of v_i and $q_i' < q_{i-1}$, for $i = k-1,\ k-2, \cdots, 1$. Set $P_i := v_0 v_1 \cdots v_{i-1} v_i'$.

Step 2 If no such point is found, set $P_i := v_0 v_1 \cdots v_i v_i^* \hat{v}$ where v_i^* and $\hat{v}$ are the first pair of points such that $\hat{v}$ is a child of v_i^* and $\frac{1-q_i^*}{\hat{q}-q_i^*}$ is less than the projected distance[3] between the destination and v_i^* for $i = k-1,\ k-2, \cdots, 0$.

It should be noted that the path cost will increase when a backtrack point is not found in Step 1.

Another key issue associated with the efficiency of any search method is the search direction. Backward search is usually more efficient than forward search when there are more initial states (i.e., origins) than final states (i.e., goals). Note that this usual selection of search direction is not directly applicable to robot path planning, since one and only one origin–destination pair needs to be considered each time in robot path planning. However, a simple modification in accordance with the following

[2]For notational convenience, we used a single, instead of triple, subscript to represent a point or cube in the workspace.

[3]That is, the minimum number of points between v_i^* and the destination in the absence of obstacles.

observation will make the usual selection applicable to robot path planning: there are more ways to reach a point when the probability of the point meeting a deadend is small than when this probability is large. In other words, backward search is more efficient than forward search when the destination has a higher probability of meeting a deadend than the origin. Our experiments show that on the average, 30% of deadends can be avoided by exchanging[4] the origin and the destination based on their probabilities of meeting deadends.

The procedure **Construct_Path** described below constructs two paths each time: FPATH starting from the origin and BPATH from the destination. Out of these two paths, a path which is less probable to meet a deadend will be expanded first. The search will continue to reduce the differences between the two paths, FPATH and BPATH, until the two paths intersect.

Construct_Path

1. Initialize the origin with FORWARD and the destination with BACKWARD. Set FPATH and BPATH to null sets.

2. Repeat 2a-d until $FPATH \bigcap BPATH \neq \emptyset$:

   2a. Select a node which has a lower probability of meeting a deadend from FORWARD or BACKWARD. Call that node CURRENT.

   2b. Insert CURRENT to the corresponding set, FPATH or BPATH.

   2c. Choose a successor of CURRENT from its neighbors. A node which minimizes the path cost is selected as the successor. If there is more than one node with the minimal path cost, a node which has the lowest probability

[4]Switching between backward search and forward search.

Heuristic	L_1	L_2	L_∞	(1)	q_{ij}
Avg. path length	23.41	16.45	16.47	16.45	16.47
Avg. # of node created	88.54	79.00	73.28	94.23	68.76
Avg. # of node examined	123.54	117.47	124.56	138.18	115.59
MCFO	0.723	0.464	0.792	0.638	0.621

(1) : Simulation using L_2-metric while increasing clearance.

Table 5.1: Simulation results without adjusting the search direction.

of meeting a deadend is chosen as the successor. If there is no successor for CURRENT, backtrack to an earlier node.

2d. Call the successor FORWARD or BACKWARD, depending on its predecessor.

5.4 Experimental Results

The procedure **Construct_Path** was simulated in a workspace composed of 32×32 points. First, the workspace is converted to the probability map shown in Fig. 5.1 and Fig. 5.3 using the procedure **Calc_Prob**. Origin-destination pairs are generated using a uniform random number generator. As the safety measure of a path P, the *mean clearance from obstacle* is defined as:

$$MCFO(P) = \sum_{v_i \in P} \min_{v_j \in V_{obs}} d_\infty(v_i, v_j).$$

Table 5.1 shows the results obtained from depth-first search using various heuristic estimators. Each data is based on 100 origin-destination pairs. Note that there is a 10–20% increase in efficiency. Table 5.1 is obtained using the same experiments as Table 5.2 except that the search direction has been decided from the d_∞ clearance

Heuristic	L_1	L_2	L_∞	(1)	q_{ij}
Avg. path length	23.41	16.45	16.47	16.45	16.47
Avg. # of node created	80.70	74.31	68.01	94.23	61.98
Avg. # of node examined	115.22	111.04	127.44	127.44	108.21
MCFO	0.726	0.464	0.793	0.638	0.624

Table 5.2: Simulation results while adjusting search direction with clearance.

from the nearest obstacle (minimum clearance) while Table 5.3 is the result of using q_{ij} to determine the search direction. Notice that every phase of the search improves from Table 5.1 to Table 5.2 and from Table 5.2 to Table 5.3, while the quality of the chosen path remains unchanged. This shows the importance of the search direction. It also shows the superiority of q_{ij} over minimum clearance. This is because minimum clearance only provides the distance from nearest obstacle while there are many other factors to be considered. Some of the factors are the size and orientation of each obstacle, and number of obstacles within a close range. It is very difficult to consider all this information at the same time because their influence is difficult to quantify. Unlike minimum clearance, q_{ij} increases not only when the obstacle is closer but also when obstacle is larger or the number of obstacles increases.

The difference between L_1- and L_∞- metrics can be explained as the difference in metric systems. That is, any L_1-metric algorithm can be transformed into L_∞-metric algorithm [37] by transforming its coordination (x, y) with a new coordination (x', y) where

$$x' = \sqrt{2}(x + y)$$

$$y' = \sqrt{2}(x - y).$$

Heuristic	L_2	L_∞	q_{ij}
Avg. path length	16.45	16.47	16.47
Avg. # of node created	68.71	62.66	52.94
Avg. # of node examined	106.94	112.18	99.34
MCFO	0.464	0.792	0.622

Table 5.3: Simulation results while adjusting search direction with q_{ij}.

Hence, the resulting algorithm requires adjustments with a scale factor $\sqrt{2}$. Though the results are equivalent, it is better to select L_∞-metric in this example as the computation time is a major factor.

The results of our path planner for two different workspaces are compared with the optimal paths obtained using the A^* algorithm. The heuristic used in the A^* algorithm is $f(v) = g(v) + h(v)$, where $g(v)$ is the actual distance from the origin to the current point v and $h(v)$ represents an estimated cost from v to the destination. The estimated cost $h(v)$ from v to the destination is obtained as $h(v) = d_\infty(v, w) - c_\infty(v)$ where $c_\infty(v)$ is the minimum clearance of v. In [26], Kambhampati and Davis used this heuristic estimator to construct a path for a mobile robot. With limited information given in [26], it is not feasible to compare their algorithm with our path planner. Table 5.4 presents the experimental results of our path pianner and those of A^*. It was shown that our path planner improves the search time by one order of magnitude at the cost of slight increase (1–2%) in path length.

Our algorithm usually finds the shortest path when it requires no or a small number of sideway moves. However, it exhibited some difficulties when the shortest path requires many sideway moves. This is due to the lack of a general solution to

Workspace	Fig. 5.1		Fig. 5.3	
Algorithm	A1	A2	A1	A2
Avg. path length	16.47	16.41	17.14	16.58
Avg. # of node expansions	18.27	57.45	20.39	78.47
Avg. # of backtracking	0.21	NA	3.21	NA
Avg. CPU time	4.78	53.86	6.92	86.73

A1 : Result using our path planner.

A2 : A^* algorithm with L_∞ heuristic.

Table 5.4: Simulation results using the example in Figs. 5.1 and 5.3.

avoid local minima with the depth-first search.

Attempts have been made to utilize our probability measure with the best search algorithm such as the A^* algorithm. However, the simulation has not shown much improvements when the optimal path contains at least one sideway move. This can be explained as follows:

1. To ensure the optimality of the path, the A^* search has to check every deadend when the optimal path contains at least one sideway move.

2. Since our probability measure is based on the probability of leading to a dead-end when the hill-climbing method is applied, it does not reflect the probability of meeting a deadend with the A^* search.

5.5 Summary

In this chapter, we have developed an objective measure of describing the effects of obstacles on collision-free robot path planning. This measure is then used for a search

method similar to the hill-climbing method. The hill-climbing method generates solutions very fast if it does not encounter deadends. Although it is not possible to avoid deadends completely during the search, we can minimize the probability of encountering deadends based on the probability field developed here. Our simulation has shown that the probability measure is more effective than the distance measure which is often used in the potential field approach.

The main advantage of our algorithm will become more vivid if it is implemented for a 3D environment. The complexity of conventional algorithms increases dramatically when it is to be implemented for 3D (instead of 2D) problems. This is due to the way the field data is computed. In contrast to the gradient-field algorithms [63, 64, 85] which use the A^* algorithm to compute the cost of each cell, our algorithm computes the field data using the recursive definition of a cell leading to a deadend. This reduces the computational requirements considerably. Another advantage of using a recursive computation of the field data is flexibility. By its definition, our field data can be limited by a certain number of steps, k, thus computing the probability that a cell may lead to a deadend within k steps. This is quite common in real world applications where sensors have limited ranges.

CHAPTER VI

CONCLUSION

There are four major contributions in this thesis. First, it is shown that the minimum bound of the worst-case computational complexity of the ladder problem is $\Omega(n^2 \log n)$ instead of $\Omega(n^2)$ as known previously. Hence, both the algorithm RG included in this thesis and Levin and Sharir's work are indeed optimal rather than suboptimal. The second contribution is the proposed algorithm using the RG in Chapter 3. Though the RG has the same computational complexity as the previous work [40], its actual computation time can be improved significantly due to the small size of the graph. Its advantage is more visible when the workspace is crowded with obstacles separated by short distances. The third contribution is the partitioning algorithm of a discrete workspace. Though the worst-case performance of the algorithm remains the same as previous approaches, the average running time can be reduced by an order of the magnitude thanks to the grouping of points with similar characteristics. The final contribution is the introduction of a new heuristic in the probabilistic field. The probabilistic field is quite useful when the workspace partitioning results in severe fragmentation of the workspace or the information for the workspace is incomplete. Instead of relying on a graph generated *a priori*, the robot can move on the best direction based on the information gathered so far.

The notion of dominance relation is another useful tool developed in this thesis. Unlike many other approaches which view the relation between regions based on their physical locations only, the dominance relation provides not only physical adjacency but also master-slave relations based on their recti-linear visibility. This facilitates the search for a path more efficiently by removing inefficient search steps.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility polygon search and Eucledean shortest paths," *Proceedings 26th Symposium on Foundations of Computer Science*, pp. 155–164, 1985.

[2] D. H. Ballard and C. M. Brown, *Computer Vision*, Prentice Hall, 1982.

[3] S. Bonner and R. B. Kelley, "A novel representation for planning 3-d collision-free paths," *IEEE Trans. on Systems, Man, and Cyber.*, vol. 20, no. 6, pp. 1337–1351, Nov./Dec. 1990.

[4] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. on Systems, Man, and Cyber.*, vol. 19, no. 5, pp. 1179–1187, Sep./Oct. 1989.

[5] R. A. Brooks, "Planning collision-free motions for pick-and-place operations," *Int'l Journal of Robotics Research*, vol. 2, pp. 19–44, 1983.

[6] R. A. Brooks, "Solving the find-path problem by good representation of free space," *IEEE Trans. on System Man Cybernatics*, vol. 13, pp. 190–197, 1983.

[7] R. A. Brooks and T. Lozano-Perez, "A subdivision algorithm in c-space for findpath with rotation," *IEEE Trans. on System, Man, and Cybernetics*, vol. 15, pp. 224–233, Mar. 1985.

[8] S. J. Buckley, "Fast motion planning for multiple moving robots," *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 322–326, May 1989.

[9] J. F. Canny, "The complexity of robot motion planning," *The MIT Press*, 1988.

[10] L. P. Chew, "Planning the shortest path for a disc in $o(n^2 \log n)$ time," *ACM Proc. Symp. on Computational Geometry*, pp. 214–220, Jun. 1987.

[11] K. Clackson, S. Kapoor, and O. Vaidya, "Rectilinear shortest paths through polygonal obstacles in $o(n \log^2 n)$ time," *ACM Proc. Symp. Computational Geometry*, pp. 251–257, Jun. 1987.

[12] J. Davis and S. Tufekci, "A decomposition algorithm for locating a shortest path between two nodes in a network," *Networks*, vol. 12, pp. 161–172, 1982.

[13] B. Delaunay, "Sur la sphere vide," *Bull. Acad. Sci. USSR(VII)*, pp. 793–800, 1934.

[14] P. DeRezende, D. Lee, and Y. Wu, "Rectilinear shortest paths with rectangular barriers," *ACM Proc. Symp. Computational Geometry*, pp. 204–213, 1985.

[15] E. W. Dijkstra, "A note on two problems in connection with graphs," *Numerische Math.*, vol. 1, pp. 269–271, 1959.

[16] K. R. Goheen and E. R. Jefferys, "The application of alternative modelling techniques to ROV dynamics," *Proc. IEEE Intern. Conf. on Robotics and Automation*, vol. 2, pp. 1302–1309, May 1990.

[17] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, "Linear time algorithms for shortest path and visibility problems inside triangulated simple polygons," *Algorithmica*, vol. 2, pp. 209–233, 1987.

[18] L. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon," *ACM Proc. Symp. on Computational Geometry*, pp. 50–63, 1987.

[19] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE-SSC*, vol. 4-2, pp. 100–107, 1968.

[20] N. Hogan, "Impedance control: An approach to manipulation," *J. Dynamic Systems, Measurement, and Control*, vol. 107, pp. 1–24, Mar. 1985 1985.

[21] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, "On the complexity of motion planning for multiple independent objects: PSPACE hardness of the 'warehouseman's problem'," *Int. J. Rob. Res.*, vol. 3 (4), pp. 76–88, 1984.

[22] W. D. Howden, "The sofa problem," *Computer Journal*, vol. 11, pp. 299–301, Nov. 1968.

[23] C. L. Jackins and S. L. Tanimoto, "Oct-trees and their use in representing three-dimensional objects," *Proc. of Computer Graphics and Image Processing 14*, pp. 249–270, 1980.

[24] S. Jun and K. G. Shin, "A probablistic approach to collision-free robot path planning," *Proc. Int. Conf. Robotics Automation*, pp. 220–225, 1988.

[25] S. Jun and K. G. Shin, "Automatic path planning in discretized workspaces using dominance relation," *IEEE Trans. on Robotics and Automation*, pp. 342–350, June 1991.

[26] S. Kambhampati and L. S. Davis, "Multiresolution path planning for mobile robots," *IEEE Journal of Robotics and Automation*, vol. RA-2, pp. 135–145, Sep. 1986.

[27] J. A. Kangas, T. K. Kohonen, and J. T. Laaksonen, "Variants of self-organizing maps," *IEEE Trans. on Neural Networks*, vol. 1, no. 1, pp. 93–99, Mar. 1990.

[28] N. Katoh, T. Ibaraki, and H. Mine, "An efficient algorithm for k shortest simple paths," *Networks*, vol. 12, pp. 411–427, 1982.

[29] Y. Ke and J. O'Rourke, "Moving a ladder in three dimensions: upper and lower bounds," *ACM Proc. Symp. Computational Geometry*, pp. 136–146, Jun. 1987.

[30] K. Kedem and M. Sharir, "An efficient motion planning algorithm for a convex polygonal object in 2-dimensional polygonal space," Tech. Rept. 253, Courant Institute, New York University, New York, NY, Oct. 1986.

[31] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *IEEE Int'l. Conf. on Robotics and Automation*, pp. 500–505, Mar. 1985.

[32] B. K. Kim and K. G. Shin, "Minimum-time path planning for robot arms and their dynamics," *IEEE Trans. on Systems, Man, and Cyber.*, vol. SMC-15, no. 2, pp. 213–223, Mar./Apr. 1985.

[33] B. K. Kim and K. G. Shin, "Suboptimal control of industrial manipulators with a weighted minimum time-fuel criterion," *IEEE Trans. on Automatic Control*, vol. AC-30, no. 1, pp. 1–10, Jan. 1985.

[34] T. K. Kohonen, "Self-organized formation of topologically correct feature maps," *Bio. Cyb.*, vol. 43, pp. 59–69, 1982.

[35] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. of the IEEE Intern. Conf. on Robotics and Automation*, pp. 1398–1404, Washington, D.C., 1991, IEEE Computer Society.

[36] D. Kozen and C. Yap, "Algebraic cell decomposition in NC," *Proc. IEEE Symp. on Found. of Comp. Sci.*, pp. 515–521, 1985.

[37] D. T. Lee and C. K. Wong, "Voronoi diagrams in L_1-(L_∞-) metrics with 2-dimensional storage applications," *SIAM J. Comput.*, vol. 9, pp. 200–211, 1980.

[38] D. Lee and F. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, vol. 14, pp. 393–410, 1984.

[39] H. Lee, "Spatial decomposition and its manipulation," *Ph.D. Dissertation, Dept. Indust. Oper. Engr., Univ. of Michigan*, 1988.

[40] D. Leven and M. Sharir, "An efficient and simple motion planning algorithm for a ladder moving in a two-dimensional space amidst polygonal barrier," *Journ. Algorithms*, vol. 8, pp. 192–215, 1987.

[41] D. Leven and M. Sharir, "On the number of critical free contacts of a convex polygonal object moving in 2-D polygonal space," *Discrete Comput. Geometry*, no. 2, pp. 255–270, 1987.

[42] D. Leven and M. Sharir, "Planning a purely translational motion for a convex polygonal object moving in 2-D polygonal space using generalized voronoi diagram," *Discrete Computational Geometry*, no. 2, pp. 9–31, 1987.

[43] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Trans. on System, Man, and Cybernetics*, vol. 11-10, pp. 681–698, 1981.

[44] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Trans. on Comp.*, vol. 32-2, pp. 108–120, Feb. 1983.

[45] T. Lozano-Perez and M. Wesley, "An algorithm for planning collision-free paths among polyhedra obstacles," *Comm. ACM*, vol. 22-10, pp. 560–570, Oct. 1979.

[46] J. Luh and C. Campbell, "Minimum distance collision-free path planning for industrial robots with a prismatic joint," *IEEE Trans. on Automatic Control*, vol. 29, pp. 675–680, 1984.

[47] J. Luh and C. Lin, "Approximate joint trajectories for control of industrial robots along cartesian paths," *IEEE Trans. on System, Man, and Cybernetics*, vol. 14-3, pp. 444–450, 1984.

[48] V. Lumelsky, "Effect of kinematics on dynamic path planning for planar robot arms moving amidst unknown obstacles," *IEEE Journal Robotics Automation*, vol. RA-3, pp. 207–223, 1987.

[49] V. Lumelsky, "A comparative study on the path length performance of maz-searching and robot motion planning algorithms," *IEEE Trans. on Robotics and Automation*, vol. 7, no. 1, pp. 57–66, Feb. 1991.

[50] V. Lumelsky and T. Skewis, "Incorporating range sensing in the robot navigation function," *IEEE Trans. on Systems, Man, and Cyb.*, vol. SMC-20, no. 5, pp. 1058–1069, Sep./Oct. 1990.

[51] V. Lumelsky and A. Stepanov, "Path planning strategies for a point mobile automation moving amidst unknown obstacles of arbitrary shape," *Algorithmica*, vol. 2, pp. 403–430, 1987.

[52] D. McDrrmott, "Robot planning," *AI magazine*, vol. 13, no. 2, pp. 55–79, summer 1992.

[53] D. Miller and M. G. Slack, "Global symbolic maps from local navigation," in *Proc. of the Nineth National Conf. on Artificial Intelligency*, pp. 750–755, Menlo Park, Calif, 1990, American Assoc. for Artificial Intelligence.

[54] A. Mirzaian, "Channel routing in VLSI," *ACM Symp. Theory of Computing*, pp. 101–107, 1984.

[55] E. Moore, "The shortest path through a maze," *Proc. Int. Symp. Switching Th.*, pp. 285–292, 1957.

[56] H. P. Moravec and A. Elfes, "High-resolution maps from wide angle sonar," *IEEE Conf. on Robotics and Automation*, pp. 116–121, 1985.

[57] C. O'Dunlaing, M. Sharir, and C. K. Yap, "Generalized Voronoi diagrams for a ladder, I: Topological analysis," *Comm. Pure Applied Math.*, vol. 39, pp. 423–483, 1986.

[58] C. O'Dunlaing, M. Sharir, and C. K. Yap, "Generalized Voronoi diagrams for a ladder, II: Efficient construction of the diagram," *Algorithmica*, vol. 2, pp. 27–59, 1987.

[59] C. O'Dunlaing and C. Yap, "A 'retraction' method for planning the motion of a disc," *Journ. Algorithms*, no. 6, pp. 104–111, 1985.

[60] O. Ore, *Theory of Graphs*, American Math. Society, Providence, RI, 1962.

[61] J. O'Rourke, "Lower bounds on moving a ladder," Tech. Rept. 85/20, Dept. of EECS, Johns Hopkins University, Baltimore, MD, 1985.

[62] D. Payton, "Exploiting plans as resources for action," in *Proc. of the Workshop on Innovative Approaches to Planning, Scheduling, and Control*, pp. 175–180, San Mateo, Calif, 1990, Morgan Kaufmann.

[63] D. W. Payton, "Internal plans: A representation for action resources," *Robotics and Autonomous Systems*, vol. 6, pp. 89–103, 1990.

[64] D. W. Payton, J. K. Rosenblatt, and D. M. Keirsey, "Plan guided reaction," *IEEE Trans. on Systems, Man, and Cyber.*, vol. 20, no. 6, pp. 1370–1382, Nov./Dec. 1990.

[65] J. H. Reif, "Complexity of the mover's problem an generalizations," *in Proc. 29th Ann. IEEE Symp. on Found. of Comp. Sci.*, pp. 421–427, 1979.

[66] D. A. Rosenthal, *An Inquiry Driven Vision System Based on Visual and Conceptual Hierarchies*, UMI Research Press, Ann Arbor, Michigan, 1981.

[67] A. Sankaranarayanan and M. Vidyasagar, "A new path planning algorithm for moving a point object amidst unknown obstacles in a plane," in *Proc. IEEE Intern. Conf. on Robotics and Automation*, pp. 1930–1936, Cincinnati, Ohio, 1990, IEEE Computer Society.

[68] J. T. Schwartz and M. Sharir, "On the piano movers' problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Communications on Pure and Applied Mathematics*, vol. XXXVI, pp. 345–398, 1983.

[69] J. T. Schwartz and M. Sharir, "On the piano movers' problem ii. : The case of a rod moving in three-dimensional space amidst polygonal obstacles," *Communications on Pure and Applied Mathematics*, vol. XXXVII, pp. 815–848, 1984.

[70] J. Schwartz, "On the piano movers' problem : Iii. coordinating the motion of several independent bodies ; the special case of circular bodies moving amidst polygonal barriers," *IJRR*, vol. 2-3, pp. 46–75, 1983.

[71] J. Schwartz and M. Sharir, "A survey of motion planning and related geometric algorithms," in *Special Volume on Geometric Reasoning, Artificial Intelligence: An International Journal*, number 1-3, pp. 157–170. North-Holland, Amsterdam, Dec 1988.

[72] K. G. Shin and X. Cui, "Collision avoidance in a multiple-robot system using intelligent control and neural networks," *Proc. IEEE Conf. on Decision and Control*, pp. 130–135, Dec. 1991.

[73] K. G. Shin and M. E. Epstein, "Communication primitives for a distributed multi-robot system," *Proc. Int. Conf. Robotics and Automation*, pp. 910–917, Mar. 1985.

[74] K. G. Shin, M. E. Epstein, and R. A. Volz, "A module architecture for an integrated multi-robot system," Technical Report RSD-TR-10-84, Robot Systems Division, Center for Research in Integrated Manufacturing, University of Michigan, Jul. 1984.

[75] S. Y. Shin, *Visibility in the Plane and its Related Problems*, Ph.d. dissertation, Dept. Indust. Oper. Engr., Univ. of Michigan, 1986.

[76] S. Sifrony and M. Sharir, "An efficient motion planning algorithm for a rod moving in two-dimensional polygonal space," *Algorithmica*, vol. 2, pp. 367–402, 1987.

[77] P. Spirakis and K. Yap, "Strong np-hardness of moving many ($n \geq 8$) discs," *Information Processing Letters*, vol. 19, pp. 55–59, 1984.

[78] N. Sreenath, "Nonlinear control of multibody systems in shape space," *Proc. IEEE Intern. Conf. on Robotics and Automation*, vol. 3, pp. 1776–1781, May 1990.

[79] S. H. Suh and K. G. Shin, "Robot path planning with a weighed distance-safety," *Proc. 26-th Conf. on Decision and Control*, pp. 634–641, 1987.

[80] A. Tarski, *A Decision Method for Elementary Algebra and Geometry*, Univ. of Calif. Press, Berkeley, second ed. edition, 1951.

[81] C. Thorpe, "Path relaxation: Path planning for a mobile robot," *Proc. of the National Conference on Artificial Intelligence*, Aug. 1984.

[82] C. Thorpe, M. H. Hebert, T. Kanade, and S. Shafer, "Vision and navigation for the carnegie-mellon navlab," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 362–373, 1988.

[83] S. M. Udupa, "Collision detection and avoidance in computer controller manipulators," *5th Int. Joint Conf. Artificial Intelligence*, pp. 737–748, 1977.

[84] A. Yao, D. Avis, and R. Rivest, "An $\omega(n^2 \log n)$ lower bound to the shortest paths problem," *ACM Symp. Theory of Computing*, vol. 11, pp. 11–17, 1979.

[85] Y. Zhao, *Theoretical and Experimental Studies of Mobile-Robot Navigation*, Phd. thesis, Univ. of Michigan, 1991.

[86] Y. Zhao and T. E. Weymouth, "An adaptive route-guidance algorithm for intelligent vehicle-highway systems," *Proc. American Control Conference*, pp. 2568–2573, Nov. 1991.

[87] D. Zhu and J. Latombe, "New heuristic algorithms for efficient hierarchical path planning," *IEEE Trans. on Robotics and Automation*, vol. 7, No. 1, pp. 9–20, Feb. 1991.