Correspondence

Two Algorithms for a Reachability Problem in One-Dimensional Space

Dajin Wang and Klaus Sutner

***Abstract*—Two algorithms are proposed to solve a reachability problem among time-dependent obstacles in one-dimensional (1-D) space. In the first approach, the motion planning problem is reduced to a path existence problem in a directed graph. The algorithm is very simple, with running time $O(n^2)$, where n is the complexity of obstacles in space–time. The second algorithm uses a sweep-line technique and has running time $O(n \log_2 n)$. Besides, the latter algorithm can be easily modified to compute a collision-free trajectory, if such trajectories exist.**

***Index Terms*—Algorithmic robotics, motion planning, obstacles, path planning, reachability, space-time, sweep-line algorithm.**

I. Introduction

The general purpose of robot motion planning is to plan the movement of a robot in a known or unknown environment filled with obstacles such that certain requirements are met. For example, the robot might be commanded to navigate among a collection of static or time-dependent (moving) obstacles; the shapes of those obstacles may range from very simple ones (such as a single segment) to arbitrarily complicated objects. The robot has to move from an initial position to a target position without colliding with any of the obstacles. Moreover, there can be many different restrictions on the types of movement of the robot, and the robot's mechanical limitations have to be taken into account when planning its motion. For example, the motors that drive the robot will have limited top-speed/acceleration, causing the robot to navigate with speed/acceleration not exceeding certain value. For a non-disk-like robot whose workspace is two-dimensional (2-D), its movement can be either translational or rotational. Thus one possible restriction is that only translational movements are allowed. So there are many combinations of the restrictions and conditions on the moving robot. Some of them have been studied as special cases of motion planning problem since the general planning has been found to be computationally intractable. In fact, the significance of studying those "special cases" is not only that they are more tractable but also that they are of theoretical as well as practical interest in their own rights. In the past years, many efficient algorithms have been developed for special cases of motion planning. A typical example is the motion planning of a straight-line segment (a "rod") in a 2-D space filled with segment-like obstacles. An algorithm was proposed by Leven and Sharir [6] solving the problem in $O(n^2 \log_2 n)$, where n is the number of segment obstacles. Another example is to plan the motion of a circular disk in a 2-D polygonal workspace. Ó'Dunlaing and Yap [10] used the so-called retraction method to design an algorithm which solves the problem in $O(n \log_2 n)$. Some other efficient algorithms for various special case motion planning problems can be found in [2]–[4], [7]–[9], [15], and [16].

The motion planning in the presence of time-dependent obstacles came to researchers' attention relatively late. Two early papers in this field were written by Sutner and Maass [14] and Reif and Sharir [13]. Both papers showed that the general motion planning problem for time-dependent obstacles is PSPACE-hard. However, for some special cases, it is still possible to find efficient algorithms. In [14], an $O(n \log_2 n)$ algorithm was developed to solve the reachability problem in one-dimensional (1-D) space, where the obstacles are assumed to be disjoint polygons in *space–time* (the physical space plus time dimension) and n is the total number of edges of the polygons. In [5], Lee and Lee addressed the problem of collision-free motion planning for two robots. The presence of the first robot on the path of the second robot could be viewed as a time-dependent obstacle. Using the collision map technique, [5] computed an appropriate speed reduction and/or time delay for the second robot to avoid the collision.

An important problem in advanced robotics research is to decide whether a robot can reach a target, given the description of the environment. In this paper, we will present two algorithms to solve a special reachability problem in 1-D space. The results of this paper extend the previous work of [14]. The problem to be solved is stated as follows. Suppose a point-like robot intends to move from an initial position to a target position along a 1-D track. The robot is supposed to reach the target before a deadline time. However, the track is not a clear one: between the initial and target positions, there are various obstacles that are appearing and disappearing as the time passes. We ask the question that given the obstacle information, can the robot reach the target before the deadline?

There are many real-world problems that can be modeled or approximated with the above problem. For example, suppose a robot R moves between two spots of a workspace. The path of the robot is a fixed rail (so it can be modeled with 1-D space). In the same workspace, there are some other robots moving around so that they cross the rail from time to time. It is then natural to ask if R can reach its target without colliding with other robots. Furthermore, if the answer is yes, it is highly hoped that a trajectory of R be computed that reaches the target before the deadline, dodging all obstacles.

Certain simplifications are made to handle the problem efficiently. First of all, we assume that all obstacles are of rectangular shapes in space–time. The reasonableness of this assumption is twofold: 1) in the example given above, if the crossing robots have the shapes similar to rectangles (like trailers), the assumption nicely fits into the situation; 2) for a more complex-shaped obstacle, we can use a set of small rectangles to approximate it so that the algorithms developed for rectangular obstacles can be carried over to irregular obstacles. Secondly, we assume that the robot switches from one speed to another with very little time, so that we can almost view its acceleration as infinite. Although an infinite acceleration will never be possible, this assumption will gain more and more practicality and give better and better approximation results as manufacturing technology develops.

We propose two different approaches to solve the reachability problem. In the first approach, we construct a directed graph (or digraph for short) $G = (V, A)$ out of the layout of the obstacles.

Manuscript received February 5, 1993; revised April 7, 1994, February 5, 1995, and October 22, 1997.

D. Wang is with the Department of Computer Science, Montclair State University, Upper Montclair, NJ 07043 USA (e-mail: wang@pegasus.montclair.edu).

K. Sutner is with the Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213 USA.

Publisher Item Identifier S 1083-4427(98)04355-0.

A theorem is given establishing the relationship between the original reachability problem and the digraph. Using the theorem, we can solve the reachability problem by checking if there exists a directed path in the digraph that satisfies certain conditions. The construction of G and the checking of the existence of the directed path both take $O(n^2)$ steps, where n is the number of atomic obstacles in the space–time under consideration. For the purpose of easy analysis and correctness proof, the algorithm is first presented assuming that the robot can move with unlimited velocity. Once the idea of the algorithm is grasped, it is not a difficult task to adapt it to the case of noninfinite velocity. The digraph algorithm is straightforward in concept, uses no complicated techniques and therefore is easy to implement.

The second approach proposed is a complex sweep-line algorithm, which uses some earlier results in computational geometry. The total running time of the algorithm is $O(n \log_2 n)$. But to make use of the results in computational geometry to achieve the better running time, we have to make one further assumption: the robot in question can move very fast, so fast that the time it needs to move from the initial to target position is so little compared to the time-span of the obstacles that we can approximate the robot's movement in space–time with a vertical segment. With this approximation, we can not only get a faster algorithm for the reachability problem, but can also, by adding some actions into the algorithm, compute a collision-free trajectory from the initial to the target position, if such trajectories exist.

The rest of the paper is organized as follows. In Section II, we give the definitions of the technical terms and then formally state the problems we will solve. In Section III, we present the digraph algorithm to solve the reachability problem and analyze its complexity, assuming the infinite velocity of the robot for the time being. In Section IV, we extend the result of Section III to the more practical, noninfinite velocity. In Section V, a sweep-line algorithm is proposed to solve the problem and its complexity analyzed. In Section VI we modify the sweep-line algorithm to compute a collision-free trajectory, provided that such trajectories exist. We then give some concluding remarks in Section VII.

II. Terminology

The *time-dependent obstacle* can be described in two possible ways. We could first give the description of the physical shape of the obstacle, and then describe its movement. Alternatively, we can combine the above two pieces and describe the time-dependent obstacle as a point-set in *space–time*, i.e., the robot's workspace plus the time axis. In the study of motion planning with top-speed and/or acceleration constraint, it is found to be extremely convenient to express the movement of both robot and obstacle in space–time. The advantage of this method is that it can even reflect the changes in shape of the obstacles. We will express the obstacles in space–time. Therefore the complexity of obstacles will be measured as the complexity of the corresponding point-set in space–time. Notice that if we consider the movement of a robot in space–time, the irreversible movement along the time axis is always forced. So in our case, where the robot moves in 1-D space, the problem is quite different from motion planning in 2-D space. A point (x, t) in space–time is called a *location*. We denote $X(x, t) = x$, $T(x, t) = t$, and call x the robot's *position* at time t. We consider motion planning with the assumption that the complete information about obstacles—the shape, the trajectory along which they are moving, etc.—is known in advance and stored with some appropriate data structures, i.e., the sensing of the environment has been done before the robot starts navigating.

We are now ready to formally state the reachability problems we consider in this paper. Suppose we are given a point-like robot that

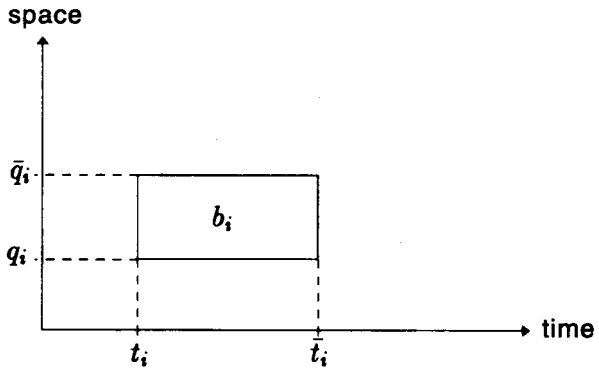
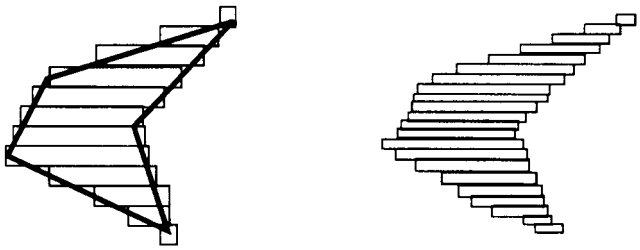


Fig. 1. An atomic rectangular obstacle in space–time.



(a) (b)

Fig. 2. (a) An approximation of an arbitrary obstacle and (b) a finer approximation of the same obstacle.

can navigate in interval $[0, M]$ in 1-D space. The robot can assume arbitrary velocity and the switching from one velocity to another takes no time.

Instance: A set of n rectangular obstacles $B = (b_1, b_2, \cdots, b_n)$ in space–time. The initial location (x_0, t_0) of the robot such that $x_0 \geq 0$. A destination X such that $x_0 \leq X \leq M$. The deadline time $T > t_0$.

Problem 1: (Reachability Problem) Can the robot get from x_0 to X before T, avoiding all obstacles?

Problem 2: (Trajectory Finding Problem) If the answer to Problem 1 is yes, specify one such trajectory.

A rectangular obstacle, which can be specified by four parameters $[t_i, \bar{t}_i, q_i, \bar{q}_i]$, represents the "blockade" of a portion of the space for a certain amount of time. We define $\mathcal{T}(b_i)$ to be projection of the obstacle b_i on the time axis, i.e., $\mathcal{T}(b_i) = [t_i, \bar{t}_i]$. See Fig. 1 for the illustration of such an obstacle.

An arbitrarily shaped obstacle in space–time can be approximated by a set of rectangular obstacles, such as shown in Fig. 2. The accuracy of approximation will depend on how many "small rectangles" are used to imitate the original obstacle—the smaller and the more rectangles, the better imitation, but more running time is needed.

III. The Directed Graph Algorithm

This approach solves problem 1 with a directed graph (digraph for short) constructed out of the layout of the obstacles. Given rectangular obstacle set $B = (b_1, b_2, \cdots, b_n)$ in space–time, construct a digraph $G = (V, A)$ with the set of vertices $V = \{v_1, v_2, \cdots, v_n\}$, where vertex v_i represents obstacle b_i. The set of edges A is defined as follows:

$$A = \{(v_i, v_j), (v_j, v_i) | b_i \cap b_j \neq \emptyset\} \bigcup \{(v_i, v_j) | (b_i \cap b_j = \emptyset) \wedge (\bar{q}_i \leq q_j) \wedge (t_j < \bar{t}_i < \bar{t}_j)\}.$$

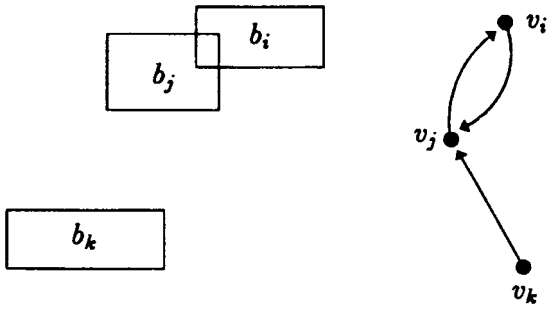


Fig. 3. Three obstacles and the corresponding digraph.

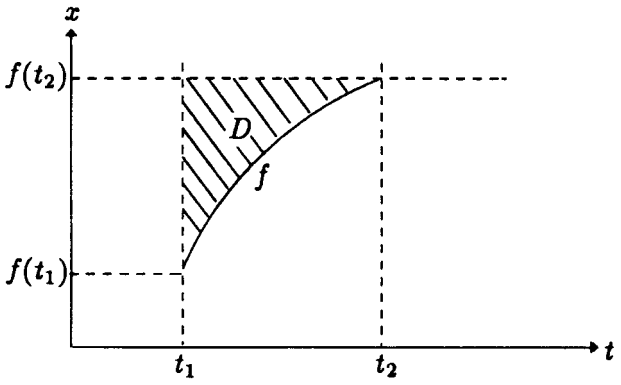


Fig. 4. Region D.

Fig. 3 gives an example of a set of obstacles and the corresponding digraph.

In the following discussion, we will establish the relationship between two different kinds of paths. One is the physical path in space–time from the source x_0 to the destination X without hitting any obstacle. The other is the combinatorial path in the digraph. To avoid confusion, we will call the former a *safe trajectory*.

We are interested in a special type of safe trajectory, on which the robot either stands still, or moves with maximum/minimum velocity. We call a safe trajectory of this type a *bang-bang trajectory*.

Lemma 1: If there exists a safe trajectory P_1, there must exist a bang-bang trajectory P_2.

Proof: If there exists a safe trajectory, then we can partition it into portions such that each portion is monotone. We will only show how to replace a monotonically increasing portion with a bang-bang trajectory.

Suppose the time projection of a monotonically increasing portion is $[t_1, t_2]$, and call this portion f. Let D be the region bounded by the following three curves:

- f;
- The vertical line passing through $[t_1, f(t_1)]$;
- The horizontal line passing through $[t_2, f(t_2)]$.

See Fig. 4 for the illustration of D.

If there exists an obstacle b_j such that $q_j = \min\{q_i | b_i \cap D \neq \emptyset\}$, then we can change f as follows.

Starting from $[t_1, f(t_1)]$, go forward with maximum velocity ($+\infty$ here) until the horizontal line $x = q_j$ is hit. Then stand still until f is reached. Notice that we are able to do so because $x = q_j$ is the lowest obstacle boundary that intersects with D. Suppose the time of reaching f is t'. For the remaining portion of f from $[t', f(t')]$ to $[t_2, f(t_2)]$, repeat the preceding procedure until no such a q_j is found.

If there exists no more such q_j, then just go forward with full velocity until reaching the horizontal line passing $[t_2, f(t_2)]$. Then stand still to reach $[t_2, f(t_2)]$. □ For Lemma 1

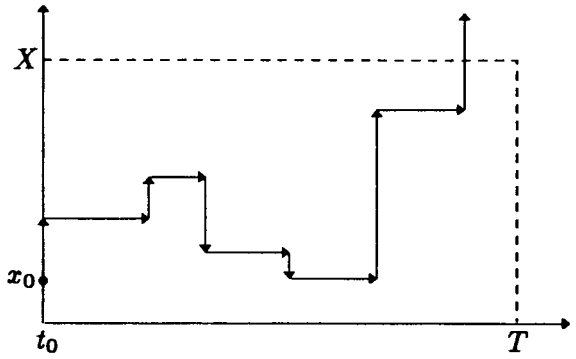


Fig. 5. A safe trajectory satisfying (a), (b), (c), and (d) of Corollary 1.

With the lemma, Corollary 1 immediately follows.

Corollary 1: If infinite velocity is allowed, and there exists a safe trajectory at all, then there is a safe trajectory which can be described as a sequence of segments $\overline{p_1, p_2}, \overline{p_2, p_3}, \cdots, \overline{p_{r-1}, p_r}$ such that

a) $X(p_1) = x_0, T(p_1) = t_0$.
b) $X(p_r) = X, T(p_r) < T$.
c) For any segment $\overline{p_i, p_{i+1}}$, either $X(p_{i+1}) = X(p_i)$ (standing still) or $T(p_{i+1}) = T(p_i)$ (moving with $\pm\infty$ velocity).
d) The projections of any two horizontal segments on time axis are disjoint (no reversal in time).

Fig. 5 shows an example of such a trajectory (bang-bang).

The following theorem establishes the relationship between the safe trajectory and the directed path in G.

Theorem 1: There exists a safe trajectory if and only if there does not exist a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that

1) $t_{i_1} \leq t_0$.
2) $q_{i_1} \geq x_0$.
3) Either i) $\overline{t}_{i_m} \geq T$, or ii) $q_{i_m} \leq 0$, or iii) $t_{i_m} \leq t_0 \wedge \overline{q}_{i_m} \leq x_0$.

The intuitive illustration of the three types of directed paths that will keep the robot from reaching X is given in Fig. 6. We provide the proof of Theorem 1 in the Appendix.

Theorem 1 allows us to transform the motion planning problem under consideration into a path existence problem in graph theory. The digraph algorithm to solve our reachability problem is sketched as follows.

Step 1: Construct the digraph $G = (V, A)$ according to the arrangement of obstacles $b_1, \cdots, b_n$.

Step 2: Find an obstacle b_i such that $q_i = \min\{q_j | t_j \leq t_0 \wedge q_j \geq x_0\}$, i.e., the obstacle directly above the robot's initial location. If there is no such an obstacle, the answer to the original question is trivially YES.

Step 3: Compute the set $V_i = \{v_k |$there is a directed path from v_i to $v_k\}$.

Step 4: Decide if there is some $v_k \in V_i$ such that b_k satisfies either 3 i) or ii) or iii) in Theorem 1. By Theorem 1, if the answer is YES, the answer to the original question is NO; if the answer is NO, the answer to the original question is YES.

A simple example illustrating the idea of the algorithm is shown in Fig. 7.

We now analyze the algorithm's running time. To construct digraph G, we basically have to check every pair of obstacles. Since there are $O(n^2)$ pairs of obstacles, Step 1 takes $O(n^2)$ operations in the worst case. Step 2 can be embedded in Step 1. For Step 3, in the worst case we will traverse all edges of G, and the number of edges is bounded by $O(n^2)$. So Step 3 takes $O(n^2)$, too. Step 4 can be effected as V_i is being computed, and each checking takes constant time. So altogether the complexity of the algorithm is $O(n^2)$.

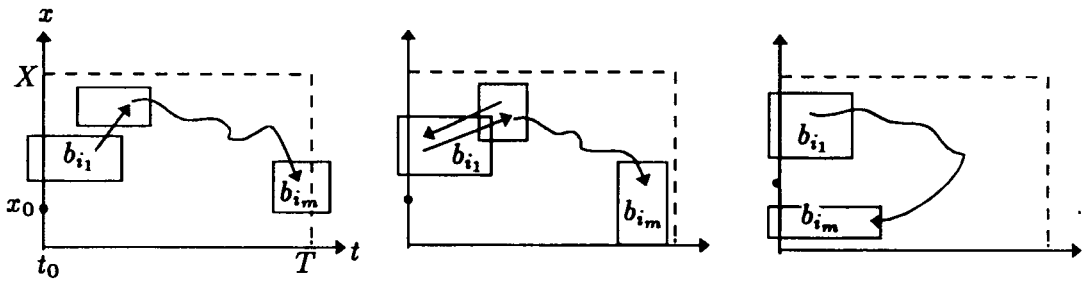


Fig. 6. The three types of directed path that will prevent the robot from reaching X.

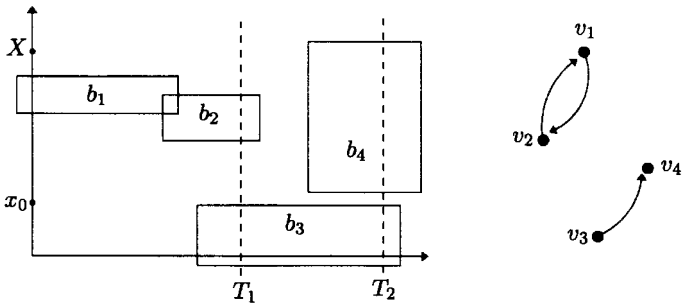


Fig. 7. If the deadline is T_1, then there exists a directed path satisfying conditions of Theorem 1 ($v_1 \longrightarrow v_2$), therefore X is not reachable before T_1. If the deadline is T_2, then there is no path satisfying conditions of Theorem 1, therefore X can be reached before T_2.

IV. The Digraph Algorithm Extended to Noninfinite Velocity

We now present the modified version of the digraph algorithm in Section III so that it can solve the reachability problem for the case of noninfinite velocity, which is a more practical assumption. The adaptation from infinite velocity to noninfinite velocity is not a difficult task. Basically, what we have to do is to change the way the digraph is constructed. Notice that a full-speed move is now a segment with certain slope, instead of a vertical segment, in space–time. As soon as the digraph is built accordingly, the reachability problem again becomes the matter of checking the existence of a directed path satisfying conditions similar to those of Theorem 1.

Given rectangular obstacle set $B = (b_1, b_2, \cdots, b_n)$ in space–time, we construct a digraph $G = (V, A)$, taking into account that the point-like robot's velocity is bounded by a constant c such that $c < \infty$. Let $V = \{v_1, v_2, \cdots, v_n\}$, where vertex v_i represents obstacle b_i. The set of edges A is defined as follows:

1) $(v_i, v_j) \in A$ and $(v_j, v_i) \in A$ if b_i and b_j intersect;
2) $(v_i, v_j) \in A$ if the following situation holds:

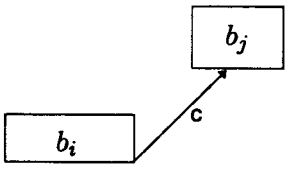


Fig. 8 gives an example of a set of obstacles and the corresponding digraph. We point out that when $c \to \infty$, the definition of edge-set A here will reduce to the same edge-set definition in Section III.

The existence of a safe trajectory before deadline is equivalent to the nonexistence of a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that

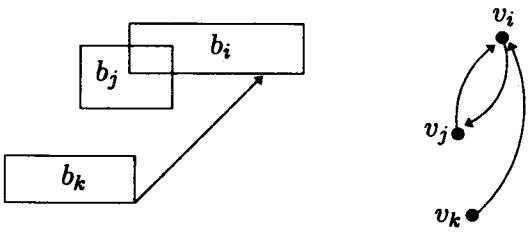


Fig. 8. Set of obstacles and the corresponding digraph.

1) starting from (x_0, t_0), the robot moving with velocity $+c$ will hit b_{i_1} at some location (x, t), where $x < X$ and $t < T$;
2) obstacle b_{i_m} is positioned in one of the three ways in Fig. 9.

Again, notice that when $c \to \infty$, condition 1 above will coincide with conditions 1 and 2 in Theorem 1, and condition 2 will coincide with condition 3 in Theorem 1.

Having modified the definition of digraph G, we can obtain a similar 4-step reachability algorithm as in Section III, which is given below.

Step 1: Construct the digraph $G = (V, A)$ according to the edge definition.

Step 2: Find an obstacle b_i such that starting from (x_0, t_0), the robot with velocity $+c$ will hit b_i at some location (x, t), $x < X$ and $t < T$. If there is no such an obstacle, the answer to the reachability problem is YES.

Step 3: Compute the set $V_i = \{v_k$ there is a directed path from v_i to $v_k\}$.

Step 4: Decide if there is some $v_k \in V_i$ such that b_k satisfies one of the three conditions in Fig. 9. If the answer is YES, the answer to the reachability problem is NO; if the answer is NO, the answer to the reachability problem is YES.

The complexity analysis takes the same line as that of Section III, resulting in an $O(n^2)$ running time.

V. The Sweep-line Algorithm

The sweep-line method discussed in this section will make use of the known efficient algorithms [12] that compute the *external contour* of a union of isothetic rectangles and determine the inclusion of a point in a plane subdivision.

Using the definition in [12], the external contour of a union F of isothetic rectangles is the boundary between F and the unbounded region of the plane. Fig. 10 gives an example of external contour of a union of isothetic rectangles in space–time. Notice that here the union of isothetic rectangles corresponds to our obstacle set B.

The following two theorems are from [12].

Theorem 2: The total number of edges forming the external contour of a union of n isothetic rectangles is $O(n)$.

Theorem 3: The external contour of a union of n isothetic rectangles can be constructed in optimal time $\theta(n \log_2 n)$.

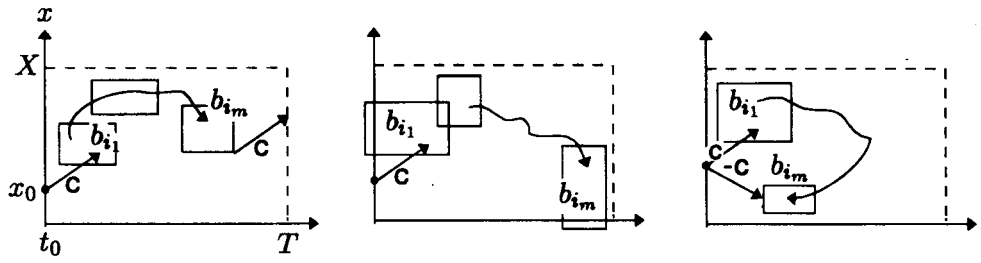


Fig. 9. The positioning of b_{i_m} that will prevent the robot from reaching the destination.

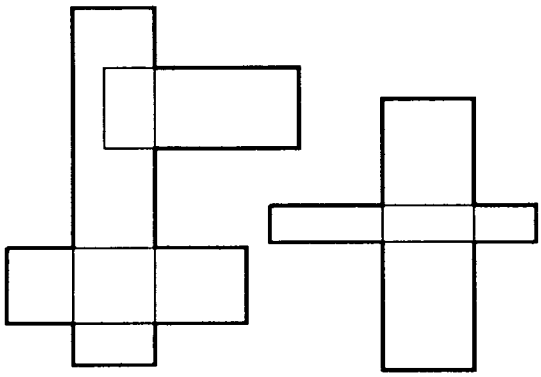


Fig. 10. The external contour of a union of isothetic rectangles.

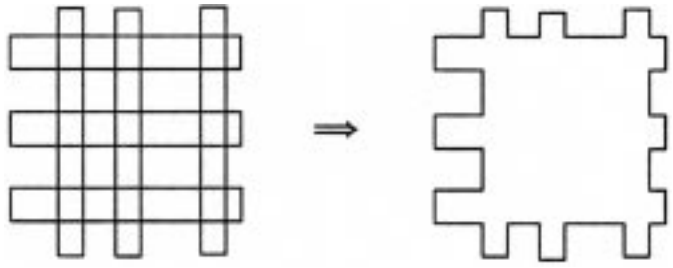


Fig. 11. After the external contour computation, many "inside" edges disappear. They do not need to be "looked at" any more.

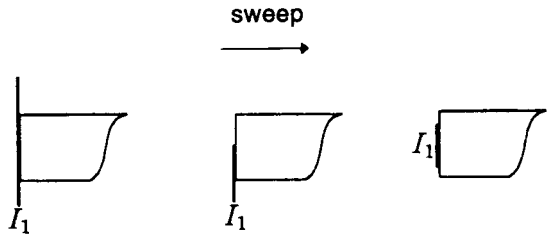


Fig. 12. (a) The edge entirely lies within I_1, (b) a portion of the edge intersects with I_1, and (c) I_1 entirely lies within the edge.

We call a group H of isothetic rectangles *connected block* if for any two points $a, b \in H$ there is a curve that entirely lies within H and connects a, b. For example, in Fig. 10 there are two connected blocks.

According to Theorem 3, we can divide $B = (b_1, b_2, \cdots, b_n)$ into several connected blocks in $O(n \log_2 n)$ steps. This result helps us develop an efficient [i.e., $O(n \log_2 n)$] algorithm to solve the reachability problem. The crucial observation that prevents us from doing lots of unnecessary work is that a connected block actually plays the role of *one* general obstacle. The boundary of that general obstacle is just the external contour of the group of related rectangular obstacles—all corners or edges "inside" the block do not have to be considered when determining the reachability.

The rest of this section will be devoted to describing the sweep-line method. The data structures used in the algorithm will be presented at appropriate stages on an as-needed basis.

Preprocessing Step 1: First of all, for the given obstacle set $B = (b_1, b_2, \cdots, b_n)$, compute the external contour. The result will be several disjoint connected blocks. Suppose there are m blocks and let B_i $(1 \leq i \leq m)$ denote the set of obstacles belonging to block i, then $B = B_1 \cup B_2 \cup \cdots \cup B_m$ and $B_i \cap B_j = \emptyset$ if $i \neq j$. Each block is now viewed as one general obstacle.

Fig. 11 illustrates the situations before and after the external contour computation. See [12] for the details of computing external contour.

Our algorithm sweeps all blocks from left to right. The time components of all *vertical* edges of the blocks constitute the set of so called *critical moments*. Each critical moment t has three tags as follows:

1) $t \uparrow block$, indicating the block it belongs to;
2) $t \uparrow ends$, the two end points of the corresponding vertical edge;
3) $t \uparrow lr$, whether the corresponding edge is a *left* one, by which we mean an edge such that the block is on its right, or a *right* one, by which we mean an edge such that the block is on its left.

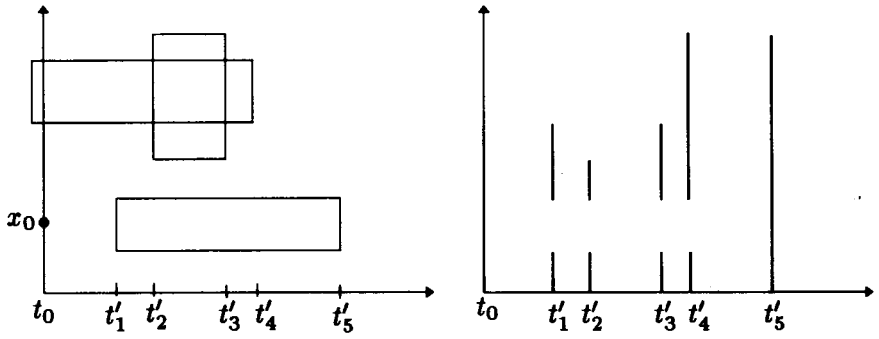
Notice that sweeping from left to right, the sweep-line "sweeps into" the block when encountering a left edge, and it "sweeps out of" the block from a right edge. Also notice that the above three pieces of information can be obtained during the computation of the external contour.

It is not difficult to realize that only at those critical moments does the reachable space-intervals of the robot change. This is the very crucial observation that makes the sweep-line method work.

Preprocessing Step 2: Sort all critical moments in nondescending order, i.e., from left to right. Denote the resulting critical-moment-list $(t'_1, t'_2, \cdots, t'_p)$ such that $t_0 \leq t'_1 \leq t'_2 \leq \cdots \leq t'_p$. All critical moments less than t_0 are ignored because the sweep starts at t_0. If all critical moments are greater than t_0, then the robot can trivially reach the target by going straight up (with infinite speed).

We need to build a data structure which we call *space–structure*. It is basically an *ordered* set of reachable space–intervals at a given critical moment. The order can be, say, from top to bottom. The sweeping algorithm is nothing more than a process of updating of the space–structure. To search, add and delete efficiently, we can resort to any balanced binary search tree [1], such as AVL tree, to implement the space–structure.

Step 1: We start the sweep from t_0 at which the only reachable space–interval is the interval between the obstacles directly above and below x_0, the robot's initial position. Clearly, these two special obstacles can be found with one scan of B. On completion of this



(a) (b)

Fig. 13. (a) The obstacles and initial position x_0 and (b) the reachable intervals at $t'_1, \cdots, t'_5$.

initial sweep step, the space–structure contains only one interval denoted as, say, I_1.

Step 2: Now consider critical moment t'_1. We modify current space–structure according to the values of t'_1 tags. By checking $t'_1 \uparrow ends$, we know if the edge intersects with I_1 or not. If not, do nothing. If the edge intersects with I_1, our updating action depends on the value of $t'_1 \uparrow lr$.

Case 1: $t'_1 \uparrow lr = left$. Then I_1 is either chopped into two shorter intervals, or cut at one end, resulting one shorter interval, or I_1 is totally gone. The situation is illustrated in Fig. 12.

Case 2: $t'_1 \uparrow lr = right$. Then I_1 gets extended at one end. To update space–structure, we need to know how long I_1 is extended. Without loss of generality, suppose I_1 should be up-extended, we then want to locate the horizontal edge of some block which is directly above I_1. Efficiently finding that horizontal edge is not trivial. However, resorting to an algorithm of locating a point in a *planar straight-line graph*, the horizontal edge can be found in $O(\log_2 n)$ with $O(n \log_2 n)$ preprocessing.

A planar straight-line graph is a graph on plane whose vertices are geometrical points, whose edges are straight lines with no intersections. It determines in general a polygonal subdivision of the plane. A *point-location* algorithm is to find the polygon in which a given point lies. An algorithm called *trapezoid method*, due to Preparata [11], can locate the query point in $O(\log_2 n)$ with $O(n \log_2 n)$ preprocessing, where n is the number of vertices of the planar straight-line graph.

Theorem 4 (Preparata/Shamos): A point can be located in an n-vertex planar subdivision with fewer than $4 \log_2 n$ tests, using $O(n \log_2 n)$ preprocessing time.

The preprocessing in the theorem corresponds to the buildup of a balanced binary search tree. Readers who are interested in knowing the algorithm in detail are referred to [11] or [12].

Now consider the collection of blocks from Preprocessing Step 1. It can be seen that those blocks also perfectly determine a subdivision of the plane. So we can construct a binary search tree out of them. Denote the binary search tree T_{trapi}. According to Theorem 4, T_{trapi} can be constructed in $O(n \log_2 n)$.

In Case 2 of Step 2, we want to locate the horizontal edge which is directly above I_1. For this purpose, we just search T_{trapi} to find the trapezoid in which the upper-end of I_1 lies. Once the trapezoid is found, we up-extend I_1 to its upper-horizontal edge. Again, by Theorem 4 this can be effected in $O(\log_2 n)$.

The actions in all steps henceforth are alike. Each of them is for one critical moment. Recall that there are p critical moments and $p = O(n)$.

Step i, $i = 3, \cdots, p$: At critical moment t'_i, we first examine $t'_i \uparrow ends$ to decide the set of reachable intervals that intersect the encountered edge. Notice that the set of intersecting intervals is a *consecutive* portion of the space–structure, by which we mean that any interval not belonging to that set is either above or below all the intervals in that set.

Without loss of generality, suppose the intersecting reachable intervals are $I_j, I_{j+1}, \cdots, I_k$, where I_j is above $I_{j+1}, \cdots, I_{k-1}$ is above I_k. Then we have $\max\{t'_i \uparrow ends\} \in I_j$, $\min\{t'_i \uparrow ends\} \in I_k$ and $I_{j+1}, \cdots, I_{k-1} \subseteq [\max\{t'_i \uparrow ends\}, \min\{t'_i \uparrow ends\}]$. Again, the updating action depends on $t'_i \uparrow lr$.

Case 1: $t'_i \uparrow lr = left$. The sweep-line sweeps "into" a block. Intervals $I_{j+1}, \cdots, I_{k-1}$ are all gone. I_j's lower-end is updated to be $\max\{t'_i \uparrow ends\}$. I_k's upper-end is updated to be $\min\{t'_i \uparrow ends\}$.

Case 2: $t'_i \uparrow lr = right$. The sweep-line sweeps "out of" a block. There are two possibilities.

Case 2.1: Either I_j gets up-extended or I_k gets down-extended. We need to search T_{trapi} to find either the trapezoid in which the upper-end of I_j lies or the trapezoid in which the lower-end of I_k lies.

Case 2.2: Some two consecutive intervals merge into one.

A simple example of sweeping process is given in Fig. 13.

To solve reachability problem, at each critical moment, after updating space–structure, we search it to find if there is an interval containing X. If we find such an interval at some critical moment, then the answer is YES. If we cannot find such an interval even after T, the answer will be NO.

We now turn to the time analysis of the algorithm. By Theorem 3, Preprocessing Step 1 takes $O(n \log_2 n)$. Preprocessing Step 2 is sorting, hence $O(n \log_2 n)$. Step 1 takes constant time. Step 2 can be done in $O(\log_2 n)$ in the worst case (the interval gets extended) with the trapezoid tree, which is built in $O(n \log_2 n)$ time. For Step 3, $\cdots$, Step p, I_j, and I_k can be found with two searches of space–structure, therefore needs $O(\log_2 n)$ time. The updating of the AVL space–structure tree again takes $O(\log_2 n)$. Lastly, at the end of each step, we check the inclusion of X in the space–structure, another $O(\log_2 n)$. So, in summary, we have the following.

Theorem 5: The reachability question can be answered in $O(n \log_2 n)$ time using a sweep-line technique, where n is the complexity of measurement of the obstacles.

In comparison with the digraph algorithm, this method is much faster when n is large. Besides, it is a reasonable conjecture that $O(n \log_2 n)$ is the optimal running time one can possibly get for the reachability problem under consideration. However, an obvious advantage of digraph algorithm over sweep-line algorithm is that the former is easier to implement. So when n is small, the digraph algorithm is preferred.

VI. Computing a Trajectory

If the answer to the reachability problem is YES, i.e., there exist trajectories from x_0 (at time t_0) to X before T, the next natural question is how to find such a trajectory.

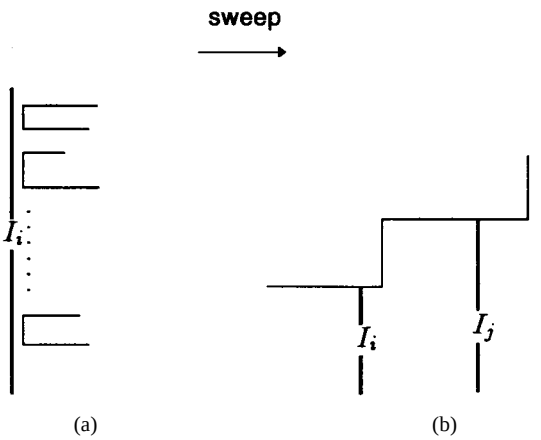


Fig. 14. (a) I_i is chopped into k pieces (k may equal to 1) and (b) I_j is obtained by expending I_i.

If the trajectory is not unique—one can easily work out a B such that there exists only one trajectory from x_0 to X—then there will be continuously many of them. That fact is easy to see: if P is a known trajectory, then any continuous change in the velocity at any moment will make a different trajectory Q. We are interested in finding just one such a trajectory, which in essence represents a family of them. For the sake of simplicity, we try to establish a *staircase* trajectory on which the robot either stands still (a horizontal line in space–time) or moves with top-speed (going straight up or down). By Lemma 1 of Section III, a trajectory of this type must exist if there exists any trajectory at all.

By embedding some actions in the sweep-line algorithm discussed in last section, we can find a staircase trajectory in $O(n \log_2 n)$ time. The basic idea of the algorithm is to construct a digraph $G_T = (V_T, A_T)$ (with T standing for trajectory) during the sweep-line process. Each vertex of G_T represents some reachable interval at a certain critical moment. An edge $(v_i, v_j) \in A_T$ only if

1) interval I_j is at the moment next to I_i's;
2) either $I_j \subset I_i$, i.e., I_j is one of the two pieces derived from chopping I_i, or $I_i \subset I_j$, which means that I_j is obtained by expending I_i.

Fig. 14 illustrates the correspondence between intervals and edges of digraph G_T. It can be seen from the figure that if there is an edge from v_i to v_j, then one can find a horizontal segment connecting I_i and I_j. Notice that the digraph G_T is constructed in *stages*, i.e., vertices of the same moment t_l can be all drawn in a vertical pattern, right to vertices of moment t_{l-1}, and left to those of t_{l+1}. Each stage represents a *distinct* critical moment and may include several critical moments (recall that critical moments $t_0 \leq t'_1 \leq t'_2 \leq \cdots \leq t'_p$, so the number of *distinct* critical moments may be less than p). All edges emanating from some vertices of a particular stage terminate at some vertices of the next stage. Therefore, in essence, all the edges of G_T direct "from left to right."

With the finished G_T, the construction of a collision-free trajectory becomes a straightforward process. We just start from vertex v_k such that I_k contains the target X and trace back until we reach the very first interval I_1, which contains x_0. The directed path leading from v_1 to v_k then perfectly represents a staircase trajectory from the original location to the target.

The important thing in this approach is to keep the size of the digraph as small as possible so that the $O(n \log_2 n)$ running time can be achieved. Naively, at each stage, we may retain vertices for all intervals of this stage. And the whole digraph G_T is staged from left to right. This construction is very clear in concept, but the total number of vertices may be $O(n^2)$, because there may be $O(n)$ reachable intervals at a stage and there may be $O(n)$ stages.

To reduce the number of vertices of digraph G_T, we manage not to retain as many as $O(n)$ vertices at every stage. Observe that at each critical moment t'_i, either one interval is chopped into two, or one interval is cut/extended at one end, or some interval(s) vanish. One strategy is then that at t'_i, we only add in new vertices that correspond to newly generated intervals. Intervals from the previous stage that don't change are left alone, with no new vertices created for them.

Each vertex has a tag of *alive/dead* indicating if the corresponding interval is still active or not. In the beginning, there is only one vertex v_1 with *alive* tag. If at the next moment, I_1 is chopped into two parts I_2 and I_3, then v_2, v_3 are added into the vertex set V_T and (v_1, v_2), (v_1, v_3) are added into the edge set A_T. The tags of new vertices are always *alive*. When an interval I_j is to be deleted from the space–structure, we change the tag of v_j to *dead*. Notice that no edge emanates from a dead vertex.

If a collision free trajectory exists at all, then at some moment we will detect, in space–structure, an active interval I_k that contains X. We then find the vertex v_k, which must be alive, and start a trace-back process from it.

Constructing the digraph $G_T = (V_T, A_T)$ as described above, G_T's size is drastically reduced because only two vertices at most are to be added to V_T for each critical moment. Since there are $O(n)$ critical moments, the cardinality of V_T is linear. The number of edges in A_T is also of order $O(n)$, because each vertex's in-degree is always 1. Since the construction of G_T can be entirely embedded in the process of making space–structure, no extra work is needed to find a specific collision-free trajectory. Thus, we have the following.

Theorem 6: If there exist collision-free trajectories from (x_0, t_0) to (X, t'), some $t_0 \leq t' \leq T$, then one can construct a staircase collision-free trajectory in $O(n \log_2 n)$ time.

VII. Conclusion

We have proposed two algorithms—the digraph algorithm and the sweep-line algorithm—to solve a reachability problem in 1-D space. The space–time plane was used as a basic tool to solve our problem. The digraph approach has its appeal due to its simplicity and easy implementation. Two versions of the algorithm were presented, for unlimited and limited robot velocity, respectively. To adapt from unlimited to limited velocity, all we have to do is to change the way the digraph is constructed. As soon as the digraph is adequately constructed, the reachability problem becomes the matter of checking the existence of a special directed path. The construction of the digraph and the checking of the existence of the directed path take $O(n^2)$ steps, where n is the complexity of obstacles. The sweep-line algorithm improves the running time of an earlier proposed algorithm for a similar problem [14] in the sense that the algorithm presented here gains $O(n \log n)$ running time for arbitrary obstacle layout, whereas in [14] to have $O(n \log n)$ running time the obstacles have to be disjoint. Yet it has to be pointed out again that our sweep-line algorithm assumes very high velocity and acceleration relative to the time-span of obstacles, so that the movement of robot can be approximately expressed with vertical segments in space–time.

Taking into account the robot's acceleration constraint turns out to be much more challenging than only considering bounded velocity. However, the literature in the past few years has already seen work done in this direction. One can easily anticipate the intractability of acceleration constrained motion planning in general terms. So most contributions have concentrated on solving special case problems, for example in [7], [16].

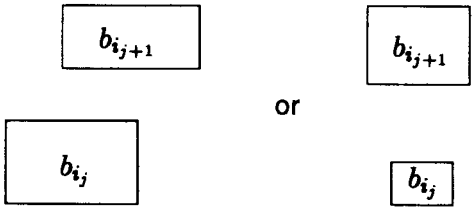


Fig. 15. $(\overline{q}_{i_j} < q_{i_{j+1}}) \wedge (t_{i_{j+1}} < \overline{t}_{i_j} < \overline{t}_{i_{j+1}})$.

APPENDIX

Proof of Theorem 1: We first show that if there exists a safe trajectory, then there does not exist a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that 1), 2), and 3) in Theorem 1 can hold simultaneously. We then show that if there does not exist a safe trajectory, then there must exist a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that 1), 2), and 3) in Theorem 1 are satisfied simultaneously.

Suppose there exists a safe trajectory. Then by Corollary 1, there is a bang-bang trajectory along which the robot either moves straight up/down or stands still. Without loss of generality, suppose the trajectory in Fig. 5 is what we have constructed. Observe the rectangular region bounded by $t = t_0$, $t = T$, $x = 0$, and $x = X$. Denote this region as S. One can see that a safe trajectory partitions S into two parts. Define

$$U = \{s \in S | \exists i [T(s) \in [T(p_i), T(p_{i+1})] \wedge X(p_{i+1}) \leq X(s)]\},$$

and

$$L = S - U.$$

Intuitively, U is the region "above" the safe trajectory (including the trajectory itself) whereas L the region "below" it. Clearly $U \cap L = \emptyset$.

Notice that if there exists a safe trajectory satisfying a), b), c), and d) in the corollary, then there will be no obstacle intersecting with the trajectory, i.e.,

$$\forall i \quad [\text{either} \quad b_i \subseteq U \text{ or } b_i \subseteq L].$$

We are now ready to show that with existence of a safe trajectory satisfying a), b), c), and d) in the corollary, there cannot be a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that 1), 2), and 3) in the theorem hold simultaneously.

For the sake of contradiction, assume there exists a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that 1), 2), and 3) in Theorem 1 hold at the same time. Then because of 1) and 2), it must be the case that $b_{i_1} \subseteq U$. Consider b_{i_m} now. If $b_{i_m} \subseteq U$, too, then neither 3(i) nor (ii), nor (iii) can hold. Therefore $b_{i_m} \subseteq L$. So there must exist $(v_{i_j}, v_{i_{j+1}}) \in A$ such that

$$(1 \leq j < m) \wedge (b_{i_j} \subseteq U) \wedge (b_{i_{j+1}} \subseteq L).$$

Since $U \cap L = \emptyset$, we have $b_{i_j} \cap b_{i_{j+1}} = \emptyset$. So from the definition of G

$$(\overline{q}_{i_j} < q_{i_{j+1}}) \wedge [\mathcal{T}(b_{i_j}) \cap \mathcal{T}(b_{i_{j+1}}) \neq \emptyset].$$

The corresponding situations are shown in Fig. 15.

One can immediately see that the situation in Fig. 15 cannot happen unless d) is violated, which means "going backward" in time—a contradiction.

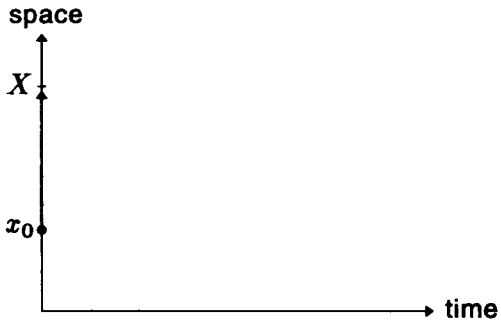


Fig. 16. There must be at least one obstacle blocking $\overline{x_0 X}$.

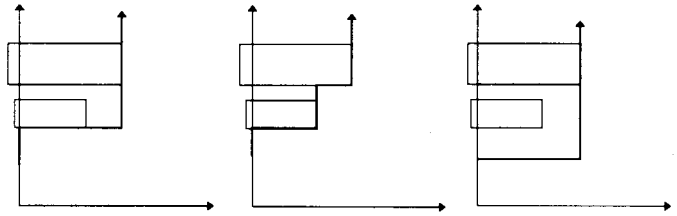


Fig. 17. The trajectory in (b) is a forward-trajectory.

We now show that if there does not exist a safe trajectory, then there must exist a directed path $(v_{i_1}, v_{i_2}, \cdots, v_{i_m})$ in G such that 1), 2), and 3) in the theorem are satisfied simultaneously.

By our assumption, there is no safe trajectory. So the vertical segment $\overline{x_0 X}$ must have at least one obstacle intersecting with it. See Fig. 16.

Let S_1 be the set of all obstacles intersecting with $\overline{x_0 X}$. The following fact is trivially true:

$$b_k \in S_1 \text{ implies that } v_k \text{ satisfies 1) and 2) in Theorem 1.}$$

If there exists a $b_k \in S_1$ such that b_k satisfies 3), then we have already constructed the required directed path (v_k), and the proof is complete.

Suppose, however, that no $b_k \in S_1$ satisfies 3., then it is obvious to see that in the presence of S_1, we can construct a safe trajectory somehow. For our purpose, we are particularly interested in one particular way of constructing the safe trajectory. As has been pointed out, there are only three types of movement of the robot in a bang-bang trajectory: going up, going down, or going right (standing still). We construct the safe trajectory in such a way that at any location-time, going-up movement has the highest priority, going-right has the second, and going-down has the lowest. In other words, the robot always tries to go upward first. It stands still only when it cannot go up any more (so it has to "drift" along the bottom of an obstacle). And it will go downward only if it cannot even stand still. In Fig. 17, only the trajectory in (b) meets this criterion. We will call this type of trajectory a *forward-trajectory* because it *goes forward as soon as it can do so.*

Now suppose that, in the presence of S_1, we have constructed the forward-trajectory F_1. From the assumption that there exists no safe trajectory, we know that there must be at least one obstacle intersecting with F_1. Let S_2 be the set of all obstacles intersecting with F_1.

Claim 1: For each $b_{i_2} \in S_2$, there is some $b_{i_1} \in S_1$ such that $(v_{i_1}, v_{i_2}) \in A$.

Proof: For the sake of contradiction, suppose the claim is not true. Then there will be some $b_{i_2} \in S_2$ such that $\forall b_{i_1} [b_{i_1} \in S_1 \longrightarrow (v_{i_1}, v_{i_2}) \notin A]$.

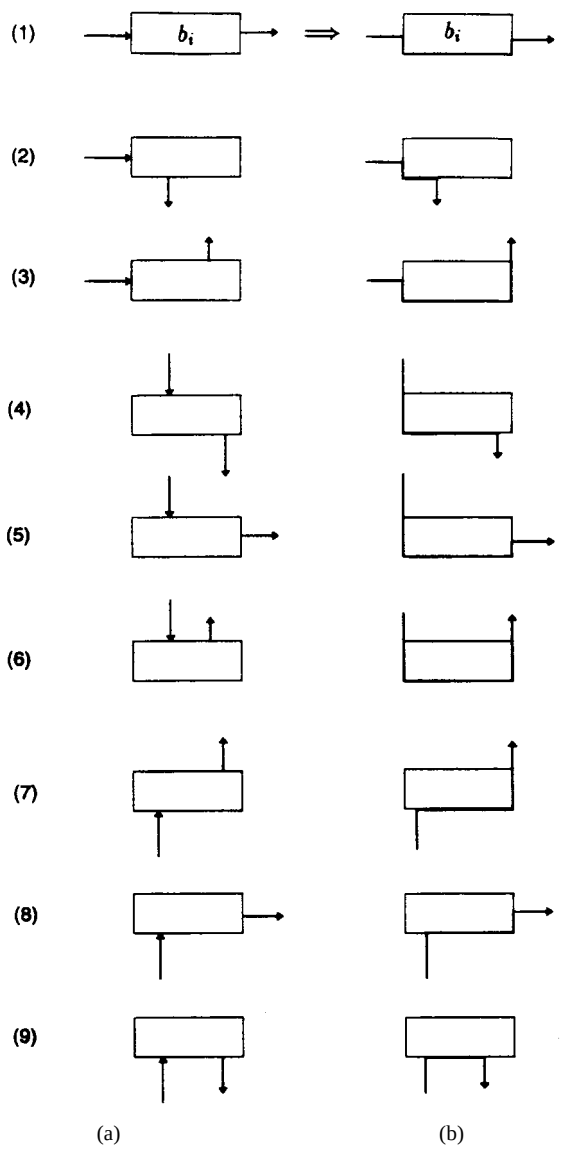


Fig. 18. (a) All possible intersection patterns and (b) the corresponding modification strategies.

An obstacle in S_2 can intersect with the current forward-trajectory F_1 in four general manners shown in (1), (3), (7) and (8) of Fig. 18(a). Since $(v_{i_1}, v_{i_2}) \notin A$, according to the definition of G, for each $b_{i_1} \in S_1$, either b_{i_1} is "above" (but doesn't intersect with) b_{i_2} or $\mathcal{T}(b_{i_1}) \cap \mathcal{T}(b_{i_2}) = \emptyset$. Then it can be seen that F_1, constructed in presence of S_1, was not a forward-trajectory. A contradiction.

□ For Claim 1

If there is a $b_{i_2} \in S_2$ such that v_{i_2} satisfies 3), then by Claim 1 we have already found the required directed path (v_{i_1}, v_{i_2}) and the proof is done. If there is no such a b_{i_2}, we will show that we can modify F_1 into a new trajectory F_2 to avoid obstacles in S_2 in a way such that F_2 remains to be a forward-trajectory. The process of modification is as follows.

Go along the current forward-trajectory F_1 until the first blocking obstacle in S_2 is encountered. The manners of the intersection of F_1 and an obstacle fall into four general situations as stated in Claim 1. We then can modify F_1 according to (1), (3), (7) and (8) of Fig. 18(b), respectively.

With F_1 partially modified, repeat the preceding procedure so that all the obstacles in S_2 are encountered and the trajectory modified accordingly. The modification will end after certain number of steps since S_2 is finite. It is important to notice that

- forward property is preserved in the modification;
- for the modification to be valid, it is necessary that b_i satisfy neither 3 i) nor ii) nor iii).

Repeatedly using the assumption that there does not exist any safe trajectory, we can inductively discriminate S_i that blocks F_{i-1}, and modify F_{i-1} into F_i, a safe forward-trajectory that avoids $S_1 \cup \cdots \cup S_i$. To adjust the current forward-trajectory F_{i-1}, observe that an obstacle in S_i intersects with F_{i-1} in nine general ways shown in (1)–(9) of Fig. 18(a). Clearly, these nine types exhaust *all* possible intersection patterns. The corresponding modification strategies are shown in Fig. 18(b).

By the two facts that

1) the number of obstacles is finite;
2) $S_i \cap S_j = \emptyset$ if $i \neq j$;

we will eventually encounter some m such that $b_{i_m} \in S_m$ satisfies 3) of the theorem.

The following lemma is just a generalization of Claim 1.

Lemma 2: For each $b_{i_{d+1}} \in S_{d+1}$, $1 \leq d < m$, there exists some $b_{i_d} \in S_d$ such that $(v_{i_d}, v_{i_{d+1}}) \in A$.

The proof of Lemma 2 is similar to that of Claim 1.

Now by Lemma 2, there are some $b_{i_{m-1}} \in S_{m-1}$, some $b_{i_{m-2}} \in S_{m-2} \cdots$ such that $(v_{i_{m-1}}, v_{i_m}) \in A$, $(v_{i_{m-2}}, v_{i_{m-1}}) \in A$, $\cdots$, $(v_{i_1}, v_{i_2}) \in A$, hence the required directed path.

□ For Theorem 1

REFERENCES

[1] G. M. Adel'son-Vel'skii and Y. M. Landis, "An algorithm for the organization of information," *Soviet Math. Dokl.*, vol. 3, pp. 1259–1262, 1962.
[2] B. K. Bhattacharya and J. Zorbas, "Solving the two-dimensional findpath problem using a line-trangle representation of the robot," Tech. Rep., School Comput. Sci., Simon Fraser University, Burnaby, BC, Canada.
[3] K. Kedem and M. Sharir, "An efficient algorithm for planning collision-free translational motion of a convex polygonal object in two-dimensional space amidst polygonal obstacles," *Proc. ACM Symp. Comput. Geom.*, 1985, pp. 75–80.
[4] K. Kedem, R. Livne, J. Pach, and M. Sharir, "On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles," *Discr. Comput. Geom. 1*, pp. 59–71, 1986.
[5] B. H. Lee and C. S. G. Lee, "Collision-free motion planning of two robots," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-17, no. 1, pp. 21–32, 1987.
[6] D. Leven and M. Sharir, "An efficient and simple motion-planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers," *J. Algorithms*, vol. 8, pp. 192–215, 1987.
[7] C. ÓDúnlaing, "Motion planning with inertial constraints," *Algorithmica*, vol. 2, no. 4, pp. 431–475, 1987.
[8] C. ÓDúnlaing, M. Sharir, and C. Yap, "Generalized Voronoi diagrams for a ladder: I. Topological analysis," *Commun. Pure Appl. Math.*, vol. 39, pp. 423–483, 1986.
[9] ——, "Generalized Voronoi diagrams for a ladder: II. Efficient construction of the diagram," Tech. Rep. 140, Comput. Sci. Dept., New York Univ., Nov. 1984.
[10] C. ÓDúnlaing and C. Yap, "A 'retraction' method for planning the motion of a disc," *J. Algorithms*, vol. 6, pp. 104–111, 1985.
[11] F. Preparata, "A new approach to planar point location," *SIAM J. Comput.*, vol. 10, no. 3, pp. 473–482, 1981.
[12] F. Preparata and M. Shamos, *Computational Geometry: An Introduction.* Berlin, Germany: Springer-Verlag, 1985.
[13] J. Reif and M. Sharir, "Motion planning in the presence of moving obstacles," in *Proc. 25th IEEE Symp. FOCS*, 1985, pp. 144–154.
[14] K. Sutner and W. Maass, "Motion planning among time dependent obstacles," *Acta Inform.*, vol. 26, pp. 93–122, 1988.
[15] S. Sifrony and M. Sharir, "An efficient motion planning algorithm for a rod moving in two-dimensional polygonal space," Tech. Rep. 8540, Eskenasy Inst. Comput. Sci., Tel Aviv Univ., Tel Aviv, Israel, Aug. 1985.
[16] D. Wang, "Finding the safest one-dimensional path among obstacles for the acceleration constrained robot," in *Computer Science 2: Research and Applications*, R. Baeza-Yates, Ed. New York: Plenum, 1994.