Planning the Shortest Safe Path Amidst Unpredictably Moving Obstacles*

Jur van den Berg and Mark Overmars

Department of Information and Computing Sciences,
Universiteit Utrecht, The Netherland
{berg,markov}@cs.uu.nl

Abstract. In this paper we discuss the problem of planning safe paths amidst unpredictably moving obstacles in the plane. Given the initial positions and the maximal velocities of the moving obstacles, the regions that are possibly not collision-free are modeled by discs that grow over time. We present an approach to compute the *shortest path* between two points in the plane that avoids these growing discs. The generated paths are thus guaranteed to be collision-free with respect to the moving obstacles while being executed. We created a fast implementation that is capable of planning paths amidst many growing discs within milliseconds.

1 Introduction

An important challenge in robotics is motion planning in dynamic environments. That is, planning a path for a robot from a start location to a goal location that avoids collisions with the moving obstacles. In many cases the motions of the moving obstacles are not known beforehand, so often their future trajectories are estimated by extrapolating current velocities (acquired by sensors) in order to plan a path [2, 5, 10]. This path may become invalid when some obstacle changes its velocity (say at time t), so then a new path should be planned. However, there is actually no time for planning; as the world is continuously changing, the computation would already be outdated even before it is finished.

To overcome this problem, often a fixed amount of time, say τ, is reserved for planning [6, 9]. The planner then takes the expected situation of the world at time $t + \tau$ as initial world state, and the plan is executed when the time $t + \tau$ has come. This scheme carries two problems:

- The predicted situation of the world at time $t + \tau$ may differ from the actual situation when some obstacles change their velocities again during planning. This may result in invalid paths.
- The path the robot will follow between time t and time $t + \tau$ is not guaranteed to be collision-free, because this path was computed based on the old velocities of the obstacles.

* This research was supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments).

In this paper we take a first step to overcome these problems. We present an approach to compute a path from a start location to a goal location that is guaranteed to be collision-free, no matter how often the obstacles change their velocities in the future. Replanning might still be necessary from time to time, to generate trajectories with more appealing global characteristics, but the two problems identified above do not occur in our case. The first problem is solved by incorporating all the possible situations of the world at time $t+\tau$ in the world model. The second problem is solved as the computed paths are guaranteed to be collision-free regardless of what the moving obstacles do.

We assume that all obstacles and the robot are modeled as discs in the plane, and that the robot and each of the obstacles have a (known) maximum velocity. The maximum velocity of the obstacles should not exceed the maximum velocity of the robot. The problem is solved in the configuration space, that is, the radius of the robot is added to the radii of the obstacles, so that we can treat the robot as a point.

Given the initial positions of each of the obstacles, the regions of the space that are possibly not collision-free are modeled by discs that grow over time with rates corresponding to the maximal velocities of the obstacles. Our goal is to compute a *shortest path* (a minimum time path) from a start to a goal configuration that avoids these growing discs (see Fig. 1).

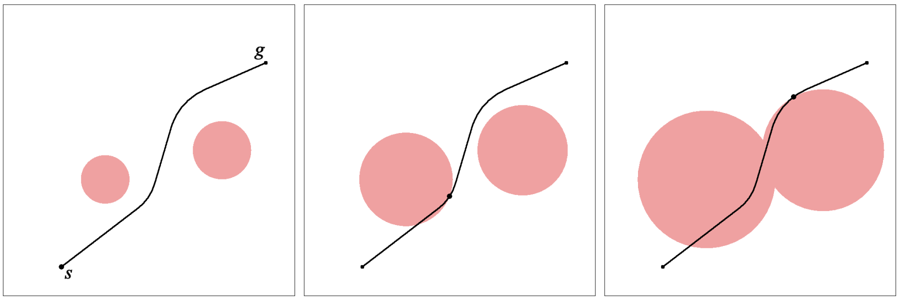

Fig. 1. An environment with two moving obstacles and a shortest path. The pictures show the growing discs at $t = 0$, $t = 1$ and $t = 2$, respectively. A small dot indicates the position along the path.

Although computing shortest paths is a well studied topic in computational geometry (see [8] for a survey), the problem we study in this paper is new. In fact, it is a three-dimensional shortest path problem, as the time accounts for an additional dimension. Such problems are NP-hard in general, yet we present an $O(n^3 \log n)$ algorithm (n being the number of discs) for our problem in the restricted case that all discs have the same growth rate.[1] In case the growth rates are different, we cannot give a time bound expressed in n. Instead, we

[1] Note that the special case of discs with zero growth rate gives a two-dimensional shortest path problem, which can be solved in $O(n^2 \log n)$ time (see e.g. [4]).

implemented a practical algorithm for this general case, that appears to work well: Experimental results show that we are able to generate shortest paths amidst many growing discs within only milliseconds of computation time.

The rest of the paper is organized as follows. We formally define the problem in Section 2. In Section 3 we examine the structure of shortest paths amidst growing discs. We sketch our global approach in Section 4, and in Section 5 we present efficient algorithms for the restricted and general case. Experimental results are given in Section 6, and Section 7 concludes the paper.

2 Problem Definition

The problem is formally defined as follows. Given are n moving obstacles $O_1, \ldots, O_n$ which are discs in the plane. The centers of the discs (i.e. the positions of the obstacles) at time $t = 0$ are given by the coordinates $p_1, \ldots, p_n \in \mathbb{R}^2$, and the radii of the discs by $r_1, \ldots, r_n \in \mathbb{R}^+$. All of the obstacles have a maximal velocity, given by $v_1, \ldots, v_n \in \mathbb{R}^+$. The robot is a point (if it is a disc, it can be treated as a point when its radius is added to the radii of the obstacles), for which a path should be found between a start configuration $s \in \mathbb{R}^2$ and a goal configuration $g \in \mathbb{R}^2$. The robot has a maximal velocity $V \in \mathbb{R}^+$ which should be larger than each of the maximal velocities of the obstacles, i.e. $(\forall i :: V > v_i)$.

As we do not assume any knowledge of the velocities and directions of motion of the moving obstacles, other than that they have a maximal velocity, the region that is guaranteed to contain all the moving obstacles at some point in time t is bounded by $\bigcup_i B(p_i, r_i + v_i t)$, where $B(p, r) \subset \mathbb{R}^2$ is an open disc centered at p with radius r. In other words, each of the moving obstacles is conservatively modeled by a disc that grows over time with a rate corresponding to its maximal velocity (see Fig. 1 for an example environment).

Definition 1. *A point $p \in \mathbb{R}^2$ is* collision-free *at time $t \in \mathbb{R}^+$ if $p \notin \bigcup_i B(p_i, r_i + v_i t)$.*

The goal is to compute the shortest possible path $\pi : [0, t_g] \to \mathbb{R}^2$ between s and g (i.e. a minimal time path with minimal t_g where $\pi(0) = s$ and $\pi(t_g) = g$) that is collision-free with respect to the growing discs for all $t \in [0, t_g]$.

3 Properties of Shortest Paths

In this section we deduce some elementary properties of shortest paths amidst growing discs. We first show that we are actually dealing with a three-dimensional path planning problem: As the discs grow over time, we can see the obstacles as cones in a three-dimensional space (see Fig. 2), where the third dimension represents the time. Each obstacle O_i transforms into a cone C_i, whose central axis is parallel to the time-axis of the coordinate frame, and intersects the xy-plane at point p_i. The maximal velocity v_i determines the opening angle of the cone, and together with the initial radius r_i, it determines the (negative) time-coordinate of the apex. The equation of cone C_i is given by:

$$C_i : (x - p_{ix})^2 + (y - p_{iy})^2 = (v_i t + r_i)^2. \tag{1}$$

The goal configuration g is transformed into a line parallel to the time-axis, where we want to arrive as soon as possible (i.e. for the lowest value of t). In the three-dimensional space it is easier to reason about the properties of shortest paths.

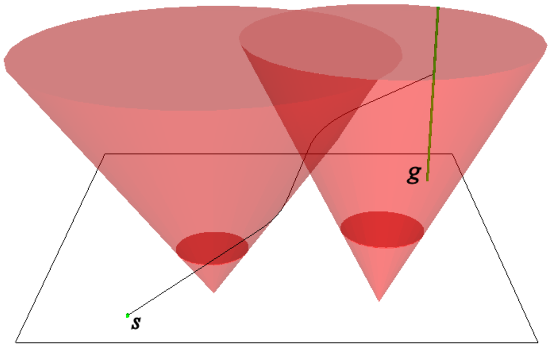

Fig. 2. The three-dimensional space of the same environment as Fig. 1

3.1 Maximal Velocity

We will first show that a shortest path is always traversed at the maximal velocity V, and hence a shortest path makes a constant angle $\arctan(1/V)$ with the xy-plane.

Lemma 1. *A point $p \in \mathbb{R}^2$ that is collision-free at time $t = t'$, is collision-free for all $t :: 0 \leq t \leq t'$.*

Proof. If $t_1 \leq t_2$, we know that $\bigcup_i B(p_i, r_i + v_i t_1) \subseteq \bigcup_i B(p_i, r_i + v_i t_2)$. Thus if a point p is collision-free at time t_2, i.e. $p \notin \bigcup_i B(p_i, r_i + v_i t_2)$, it is certainly not in $\bigcup_i B(p_i, r_i + v_i t_1)$. Hence point p is collision-free at time t_1 as well. □

Theorem 1. *The velocity $\frac{\|(\delta x, \delta y)\|}{\delta t}$ of a shortest path is constant and equal to the maximal velocity V.*

Proof. Suppose π is a path to g, of which a sub-path has a velocity smaller than V. Then this sub-path could have been traversed at maximal velocity, so that points further along the path would be reached at an earlier time. Lemma 1 proves that these points are then collision-free as well, so also g could have been reached sooner, and hence π is not a shortest path. □

3.2 Straight-Line Segments and Spiral Segments

Next, we prove that that a shortest path can only consist of straight-line motions, and motions that stay in contact with the growing discs. These latter motions follow curves 'winding' around a disc while it grows. They lie on the surface of a cone, when viewed in the three-dimensional space.

Theorem 2. *A shortest path solely consists of straight-line segments, and segments on the boundary of a growing disc.*

Proof. Theorem 1 implies that the time it takes to traverse a path is proportional to its length. Hence, parts of the path in 'open' space can always be shortcut by a straight-line segment. Only when the path stays in contact with a growing disc, it is not possible to shortcut. □

We next show that in fact, as both the velocity of the path and the growth rate of the discs are constant, the segments on the boundary of a disc are supported by a *logarithmic spiral.*

Without loss of generality, we assume that the disc has radius 0 at $t = 0$, that the disc is centered at the origin, and that the disc grows with velocity 1 (other discs can be transformed such that these conditions hold). Let the velocity of the path be V. We express the equations of the path curve in polar coordinates $(r(t), \theta(t))$, parametrized by the time t. The radius $r(t)$ of the curve at time t is equal to the radius of the disc at time t, thus:

$$r(t) = t. \tag{2}$$

The angle $\theta(t)$ is not trivially deduced, but we know that

$$\sqrt{x'(t)^2 + y'(t)^2} = V, \tag{3}$$

as the velocity along the path is constantly equal to V. From this equation, we deduce a closed form for $\theta(t)$:

$$\begin{aligned}
\{x(t) &= r(t)\cos\theta(t),\\
y(t) &= r(t)\sin\theta(t)\},\\
\{x'(t) &= r'(t)\cos\theta(t) - r(t)\theta'(t)\sin\theta(t),\\
y'(t) &= r'(t)\sin\theta(t) + r(t)\theta'(t)\cos\theta(t)\},\\
x'(t)^2 + y'(t)^2 &= r'(t)^2 + r(t)^2\theta'(t)^2 = 1 + t^2\theta'(t)^2.
\end{aligned} \tag{4}$$

Combining Equations (4) and (3), and solving for $\theta(t)$ gives:

$$\begin{aligned}
\sqrt{1 + t^2\theta'(t)^2} &= V,\\
1 + t^2\theta'(t)^2 &= V^2,\\
\theta'(t) &= \pm\frac{\sqrt{V^2-1}}{t},\\
\theta(t) &= \pm\sqrt{V^2-1}\log t + \theta_0.
\end{aligned} \tag{5}$$

Equations (2) and (5) together define a curve which is well known as the *logarithmic spiral* [11]. The $\pm$ indicates whether the spiral revolves counterclockwise $(+)$, or clockwise $(-)$ about the growing disc. The term θ_0 gives the starting angle of the spiral.

3.3 Path Smoothness

Theorem 3. *A shortest path is* C^1*-smooth.*

Proof. Suppose path π is not C^1-smooth and contains sharp turns. Then these turns could be shortcut by a straight-line segment. Hence π is not a shortest path. □

This theorem implies that in a (general) shortest path the straight-line segments and spiral segments alternate each other, and that the straight-line segments must be *tangent* to the supporting spirals of the spiral segments. In terms of the three-dimensional space this means that the straight-line segments (which "connect" two spiral segments), are *bitangent* to the cones on which the spirals lie.

3.4 Departure Curves

There are four ways in which a straight-line segment can be bitangent to a pair of cones (say C_i and C_j): left-left, right-right, left-right and right-left. In each of these cases, there is an infinite number of possible segments (whose slope corresponds to the maximal velocity V) that are tangent to both C_i and C_j. However, the possible tangency points at the surface of C_i form a continuous curve on that surface. We call such curves *departure curves.* They play a major role in our algorithm to compute a shortest path.

Definition 2. *For two cones C_i and C_j, the set $DC(C_i, C_j)$ is defined as the collection of points on the surface of C_i, for which the straight-line of slope $1/V$ that is tangent to C_i in that point is also tangent to C_j. The set $DC(C_i, C_j)$ consists of four continuous curves, each associated with one of the tangency cases. We call them* departure curves. *They are denoted $DC_{ll}(C_i, C_j)$, $DC_{lr}(C_i, C_j)$, $DC_{rl}(C_i, C_j)$ and $DC_{rr}(C_i, C_j)$, respectively.*

The set $DC(C_i, g)$ to the goal configuration g is defined similar, but then the tangent line segment should go through the goal configuration g. In this case the departure curves $DC_r(C_i, g)$ and $DC_l(C_i, g)$ are distinguished.

We now show how we can deduce equations for the departure curves on the surface of a cone C. Again, without loss of generality, we assume that the disc associated with the cone has radius 0 at $t = 0$, that the disc is centered at the origin, and that the disc grows with velocity 1. Let the velocity of the path be V. The surface of C can be parametrized by two variables, time T and angle Θ:

$$C : (T, \Theta) \rightarrow \{T \cos \Theta, T \sin \Theta, T\} .$$

Let us consider the counterclockwise spirals about this cone. Each of them is uniquely defined by the initial angle θ_0 (see Equation (5)). Each point (T, Θ) on the surface of the cone has a unique spiral that goes through that point. This spiral can be found by solving $\theta(T) = \Theta$ for θ_0:

$$\theta_0 = -\sqrt{V^2 - 1}\log T + \Theta. \tag{6}$$

Hence, the spiral though (T, Θ) is described in Euclidean coordinates as:

$$\begin{aligned}\big\{x(t) &= t\cos\big(\sqrt{V^2-1}\log t - \sqrt{V^2-1}\log T + \Theta\big),\\ y(t) &= t\sin\big(\sqrt{V^2-1}\log t - \sqrt{V^2-1}\log T + \Theta\big)\big\}.\end{aligned}$$

If we walk along this spiral, we can depart for another cone if the straight-line segment tangent to the spiral is tangent to another cone as well. The straight-line segment ℓ tangent to the spiral at point (T, Θ) is represented by:

$$\begin{aligned}\ell(t) &= \big\{x(T) + (t-T)x'(T),\ y(T) + (t-T)y'(T)\big\} = \qquad (7)\\ &= \big\{t\cos\Theta - (t-T)\sqrt{V^2-1}\sin\Theta,\ t\sin\Theta + (t-T)\sqrt{V^2-1}\cos\Theta\big\}.\end{aligned}$$

This segment must be tangent to another cone, say C_i with position p_i, initial radius r_i and velocity v_i, in order for point (T, Θ) to be on a departure curve of $DC(C, C_i)$. The surface of C_i is given by Equation (1). If we fill in line ℓ in (1), by substituting $x = \ell_x(t)$ and $y = \ell_y(t)$, and solve for t, we get a solution of the following form:

$$t_{1,2} = A(T, \Theta) \pm \sqrt{D(T, \Theta)}. \tag{8}$$

Here, $D(T, \Theta)$ is the discriminant whose sign indicates whether or not line ℓ intersects C_i. When $D(T, \Theta) = 0$, ℓ is tangent to C_i, hence $D(T, \Theta) = 0$ is an implicit equation for the set $DC_r(C, C_i)$. We can make this explicit by solving $D(T, \Theta) = 0$ for T. In Fig. 3 this function is plotted for various values of v_i (note that the function has a period of 2π). In each of these cases we see two sine-like curves (for $v_i = 1$, it is degenerate). They correspond with $DC_{rl}(C, C_i)$ and $DC_{rr}(C, C_i)$, respectively. The other departure curves $DC_{lr}(C, C_i)$ and $DC_{ll}(C, C_i)$ can be found when considering clockwise spirals.

Given a position (T, Θ) on the surface of cone C for which $D(T, \Theta) = 0$, the arrival time of the straight-line segment at cone C_i is given by $A(T, \Theta)$. The departure time of the segment is given by T. For some points along the departure curve $A(T, \Theta)$ is smaller than T. They correspond with bitangencies in the negative direction, i.e. the arrival time on C_i is smaller than the departure time at C. In the plots this is indicated by dashed curves. In the remainder of this paper these improper curves are ignored when we refer to departure curves.

We also have to take into account departure curves of $DC(C, g)$ associated with segments tangent to C and leading to the goal configuration g. In this case, we have to solve the system of equations $[\ell(t) = g]$ for T, to get a closed form for the departure curve.

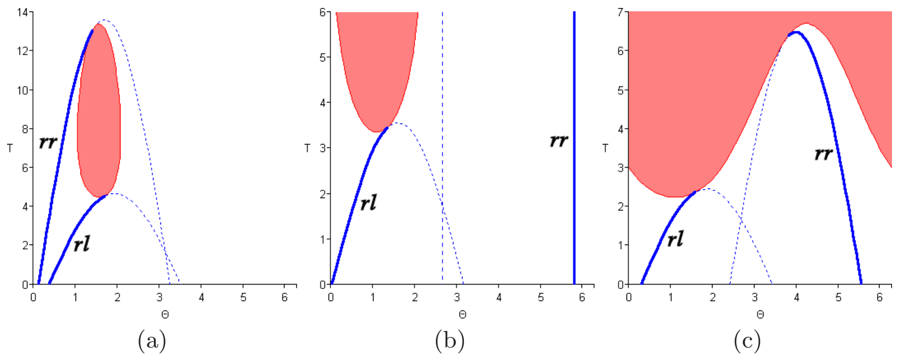

Fig. 3. Departure curves $DC_r(C, C_i)$ on the surface of C parametrized by angle Θ and time T for different values of v_i. (a) $v_i < 1$. (b) $v_i = 1$. (c) $v_i > 1$. The dashed curves are improper departure curves. The gray area, given by the inequality $(T \cos \Theta - p_{ix})^2 + (T \sin \Theta - p_{iy})^2 < (r_i + v_i T)^2$, is the region on the surface of C that is penetrated by C_i, i.e. these points are not collision-free.

4 A Naive Algorithm

With the notions introduced so far, we can devise a first, rather naive algorithm to find a shortest path amidst growing discs from some start configuration s to some goal configuration g. Our approach grows a tree of possible shortest paths that is rooted in the start configuration at time $t = 0$. A leaf is *expanded* if the length of its path from the start configuration is minimal among all leafs of the tree. To this end, each leaf is maintained in a *priority queue*, with a *key value* equal to its time coordinate (which equals the length of its path from s). The priority queue is initialized with the initial motions from the start configuration s that possibly belong to a shortest path. These are straight-line segments with slope $1/V$ leading either directly to the goal configuration, or to a tangency point on the surface of one of the cones. Some of these segments may intersect other cones, which would make them invalid, so only the collision-free segments are considered. The endpoint of each valid segment is put into the priority queue with a key corresponding to its t-value.

Now, the algorithm proceeds by handling the point with the lowest t-value in the queue (the front element of the queue). This point is either the goal configuration, in which case the shortest path has been found, or a point on the surface of a cone. In this latter, more general case we proceed by walking along a spiral about the cone. This spiral *either* runs into an obstacle (another cone), in which case there is no valid continuation of the path, *or* it encounters a departure curve on the surface of the cone. In this case there are two outgoing branches: (1) continuing along the spiral on the surface of the cone to find a next departure curve, and (2) departing for the other cone by a straight-line segment. If this latter segment is collision-free, its endpoint is inserted into the queue. Also for the first option an entry is enqueued.

This procedure is repeated until the goal configuration is popped from the priority queue. In this case the shortest path has been found, and can be read out if backpointers have been maintained during the algorithm. If the priority queue becomes empty, or if the front element of the queue has a time-value for which the goal configuration is not collision-free anymore (it is occupied by one of the growing discs), no valid path exists. In Algorithm 1, the algorithm is given in pseudocode.

Algorithm 1. ShortestPathNaive(s, g)

```
1: Initialize priority queue $\mathcal{Q}$ with endpoints of all valid outgoing segments from $s$.
2: while $\mathcal{Q}$ is not empty do
3:    Pop the front element $\langle q, t\rangle$ from the queue.
4:    if the goal configuration is not collision-free anymore at time $t$ then
5:       Path does not exist. Terminate.
6:    else if $q = g$ then
7:       Shortest path found! Terminate.
8:    else
9:       $q$ is on the surface of a cone, say $C_i$, so proceed along the spiral about $C_i$ until
         it runs into another cone, or encounters a departure curve.
10:      if the spiral encounters a departure curve, say $DC(C_i, C_j)$, then
11:         $\langle q', t'\rangle \leftarrow$ the intersection point of the spiral and the departure curve.
12:         $\langle q'', t''\rangle \leftarrow$ arrival point of the bitangent segment on the surface of $C_j$.
13:         Insert $\langle q', t'\rangle$ into $\mathcal{Q}$.
14:         if segment $\langle q', t'\rangle, \langle q'', t''\rangle$ is collision-free then
15:            Insert $\langle q'', t''\rangle$ into $\mathcal{Q}$.
16: Path does not exist.
```

In the above algorithm, we have to identify the spiral we are on (let us assume that it is a counterclockwise spiral), given a point on the surface of the cone (line 9). Let q be a point on the surface of some cone, say C_i, given in Euclidean coordinates (x, y, t). Then the corresponding coordinates (T, Θ) on the surface of C_i are given by $(T, \Theta) = (t, \arctan \frac{y - p_{iy}}{x - p_{ix}})$.

The spiral on the surface of C_i going through (T, Θ) is given by θ_0 as computed in Equation (6). Equation (5) then gives a function for the angle $\theta(t)$ along the spiral through (T, Θ). In line 10 of Algorithm 1, we wish to know whether the spiral encounters any departure curves. To this end, we should find the intersections of the spiral and the departure curves on the surface of C_i. Recall that we can deduce an implicit equation $D(T, \Theta) = 0$ for the departure curves of any pair of cones (see Equation (8)). The intersections are thus found by solving $D(t, \theta(t)) = 0$ for t.

When we have found an intersection for some value $t = T$ of the spiral and a departure curve of, say, $DC(C_i, C_j)$, we wish to know what kind of departure curve we have encountered. The arrival time at cone C_j when departed from time T is $A(T, \theta(T))$ (see Equation (8)). If this arrival time is smaller than T, the intersection can be ignored. If it is larger, we like to know whether the tangent straight-line segment arrives on the left side of C_j (and should be succeeded by a clockwise spiral on C_j), or on the right side of C_j (and should be succeeded by a counterclockwise spiral). This is determined by the derivative of $D(t, \theta(t))$ to t. If this derivative is negative at point T, we have arrived on the left side. If it is positive, we have arrived on the right side. The exact arrival location on the surface of cone C_j is given by $\ell(A(T, \theta(T))$ (see Equation (7)). From this information we can deduce the parameters defining the spiral on C_j on which we have arrived.

5 An Efficient Algorithm

The algorithm described above will indeed find a shortest path to the goal within a finite amount of time. However, in order to have a bound on the running time we must define *nodes* that can provably be visited only once in a shortest path, such that we can do *relaxation* on them as in Dijkstra's algorithm [7]. We will show that this is easy to achieve in the restricted case where all discs have equal growth rates, and present an $O(n^3 \log n)$ algorithm (n being the number of discs). For the general case this problem is left open, but we will present an algorithm that is very fast in practice, by pruning large parts of the search tree.

5.1 Discs with Equal Growth Rates

If a point $q = (T, \Theta)$ on the surface of a cone has been visited during the search for a shortest path to the goal, all points on the cone that are reachable from q by following some collision-free path with a velocity less than the maximal velocity V (i.e., $\frac{\|(\delta x, \delta y)\|}{\delta t} < V$), can never lie on a shortest path from the start to the goal (this follows directly from Theorem 1). These points are contained in the wedge formed by the clockwise and counterclockwise spiral going through q (see Fig. 4; a spiral appears as an exponential function in the ΘT-coordinate frame), as we know that on the spirals $\frac{\|(\delta x, \delta y)\|}{\delta t} = V$. We call this region the *wedge region* of q.

Let us consider the *arrangement* [1] on the surface of a cone containing all (proper) departure curves and all obstacle regions (other cones penetrating the surface) on that cone (see Fig. 5 for an impression). Note that the departure curves may be subdivided into a number of *collision-free intervals* by the obstacle regions. In case all discs have the same growth rate, say $v_i = v < V$ for all i, these intervals satisfy an interesting property (see Fig. 3(b)): let (T, Θ) be a point on some departure curve interval, then all points (T', Θ') on the same interval for which $T' > T$ are within the wedge region of (T, Θ). To prove this, we must show that for the departure curves hold that $\frac{\|(\delta x, \delta y)\|}{\delta t} \leq V$. As the proof is rather technical, we omit it here.

This means that these departure curve intervals can serve as *nodes* in our Dijkstra-algorithm. Only the path arriving earliest in an interval can contribute to a shortest path. Paths arriving later in the interval cannot be part of the shortest path, because the path arriving earliest in the interval can be extended with a traversal along the interval to end up at the same position (and time) as the path arriving later in the interval.

Each node (an interval) has two outgoing *edges*. Let the interval be a segment of a departure curve of $DC(C_i, C_j)$, then the first edge is a spiral segment to the next departure curve on the surface of C_i, and the second edge consists of a bitangent straight-line segment and a spiral segment and arrives in the first departure curve encountered on the surface of C_j. For the first edge, which stays on the cone, we have to determine the next departure curve that is encountered if we proceed by moving along the spiral about the cone. This can be done efficiently using the arrangement, if we have computed its *trapezoidal map* [1], where the sides of the trapezoids are spiral segments.

For the second edge, which traverses to another cone, we have to determine what the first departure curve is we will encounter there. This can be done efficiently using the arrangement we have computed on that cone. Using a point-location query, we can determine in what cell of the arrangement the straight-line segment has arrived, and using the trapezoidal map we know what the first departure curve is we will encounter if we proceed from there.

Finally, we must ascertain that each edge is collision-free with respect to the other cones. Spiral segments may collide with other cones if these penetrate the

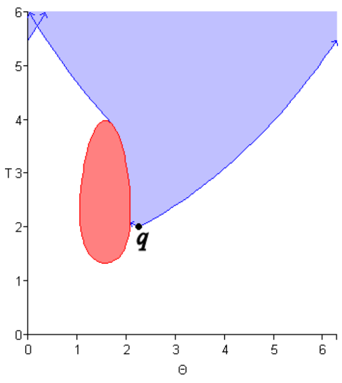

Fig. 4. The region (light grey) on the surface of a cone that is reachable from point q by paths with $\frac{\|(\delta x, \delta y)\|}{\delta t} \leq V$. The dark grey area is an obstacle.

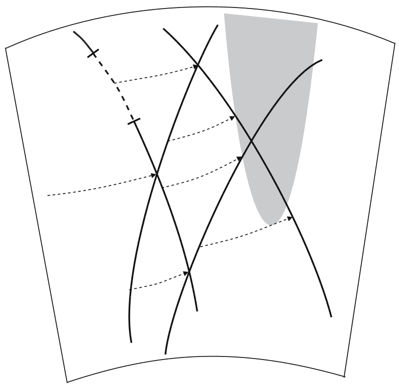

Fig. 5. An impression of an arrangement on the surface of the cone. The thick lines are the departure curves, of which one has a shadow interval (dashed). The thin dashed lines are spiral segments that delimit trapezoidal regions that have the same next departure curve or collision (only the counter-clockwise spirals are shown). The gray area depicts an obstacle area of another cone penetrating the surface, and cutting several departure curves into two intervals.

spiral's cone surface. Since obstacle areas are incorporated into the arrangement, such collisions are easily detected. Straight-line segments may collide with any cone, so for each departure curve and each cone, we calculate the *shadow interval* this cone casts on the departure curve, in which a departure will result in collision. These shadow intervals are stored in the arrangement as well. In Fig. 5, an impression is given of how such an arrangement might look.

Theorem 4. *The algorithm to compute a shortest path amidst n growing discs with equal growth rates runs in $O(n^3 \log n)$ time.*

Proof. For each pair of cones there are $O(1)$ departure curves. Since there are $O(n^2)$ pairs of cones, there are $O(n^2)$ departure curves in total. Each of the departure curves can be segmented into at most $O(n)$ intervals, as there are at most $O(n)$ cones intersecting the departure curve (each cone can split the departure curve into at most two segments). Hence, there are $O(n^3)$ departure curve intervals. Each departure curve interval has $O(1)$ outgoing edges, making a total of $O(n^3)$ edges.

The complexity of Dijkstra's algorithm is known to be $O(N \log N + E)$ where N is the number of nodes, and E the number of edges. Each edge requires some additional work. Firstly, we have to find the departure curve interval in which it will arrive, by doing a point-location query in the trapezoidal map of one of the arrangements. This takes $O(\log n)$ time. Further, we must determine whether an edge is collision-free. Using the shadow intervals stored at the departure curves, this can be done in $O(\log n)$ time as well. Thus, as both N and E are $O(n^3)$, Dijkstra's algorithm will run in $O(n^3 \log n)$ time in total.

Computing the arrangements and their trapezoidal maps takes $O(n^2)$ time per cone, as there are $O(n)$ departure curves on each cone, and $O(n)$ intersection areas of other cones. As there are $O(n)$ cones, this step takes $O(n^3)$ time in total. All the shadow intervals can be computed in $O(n^3)$ time as well, as there are $O(n^2)$ departure curves and $O(n)$ cones.

Overall, we can conclude that our algorithm runs in $O(n^3 \log n)$ time. $\square$

5.2 General Case: Discs Have Different Growth Rates

In the general case, where the discs may have different growth rates, the problem becomes much harder. We can follow the same approach as above, but let us look at what happens to the slope of the departure curves in this case (see Figs. 3(a) and (c)). In the case where the arrival cone has a slower growth rate, the departure curves (provably) satisfy $\frac{\|(\delta x, \delta y)\|}{\delta t} \leq V$ (see Fig. 3(a)). However, in the case where the arrival cone has a faster growth rate (Fig. 3(c)), it is clear that this is not the case. The departure curve DC_{rr} is horizontal at some point, meaning that $\frac{\|(\delta x, \delta y)\|}{\delta t} = \infty$. Hence, we cannot define intervals on these departure curves that serve as nodes in the search process.

We can still use Algorithm 1 for the general case, but a problem is that this algorithm considers many branches in the search tree of which we know that they will not lead to a shortest path. For instance, it lets the spirals wind around the cones forever, thereby encountering many departure curves, which in turn generate other spirals on other cones. Hence, it lets the size of the search tree blow up quickly.

In order to have an algorithm that runs fast in practice, we need to prune these useless branches of the search tree. The key observation we use for this is that a point (T, Θ) on the surface of cone C_i cannot be part of shortest path if we have visited (T', Θ) already (where $T' < T$) *and* the vertical line segment on the surface of the cone between (T', Θ) and (T, Θ) is collision-free. This is because (T, Θ) is then in the wedge region of (T', Θ) (note that the velocity $\frac{\|(\delta x, \delta y)\|}{\delta t}$ along the vertical line segment equals $v_i < V$). Hence a spiral encountering (T, Θ) need not be expanded any further.

To implement this practically, we only do this test for a constant number of Θ's. To this end, we augment Algorithm 1 by choosing a small constant ε, and drawing $\frac{2\pi}{\varepsilon}$ evenly distributed vertical lines on the surface of each cone. These vertical lines are segmented into collision-free intervals by obstacle regions on the surface. Now, these intervals will serve as *nodes* in our practical algorithm on which we perform *relaxation*.

This means that if we walk along a spiral on the surface of a cone, and the spiral crosses a vertical line, we have to check whether this spiral is the first to arrive in the particular interval. If not, this spiral can never be part of a shortest path, for the same reasons as above. Thus, this branch of the search tree can be pruned.

The smaller ε is chosen, the sooner the spirals can be pruned, and hence the smaller the size of the search tree will be. On the other hand, a smaller ε also causes the algorithm to perform more (costly) relaxation checks, with diminishing returns. So ε should not be chosen too small. Even though we are unable to bound the running time of this algorithm in terms of the number of discs (n) or the value of ε, it turns out to be very fast in practice, as we will see next.

5.3 Implementation Details

We created a fast implementation of Algorithm 1, augmented with the pruning heuristic presented above. We did not create an arrangement of all vertical lines and all obstacle regions on each cone. This would take too much time. Instead, we maintain for each vertical line m one time-value at which it was last visited, say t_m. Given the order in which the points are considered in the priority queue, we know that when a point q is popped from the queue, it has a higher time value than any point previously considered. So, if point q lies on vertical line m, its time value q_t is larger than the time-value t_m of the point previously considered on that line. If the line segment between t_m and q_t on m is collision-free, q is in a previously visited interval, and hence this point is not expanded. However, if the vertical line segment between these two points is not collision-free, point q is the first to arrive in a new interval, and its outgoing edges must be inserted into the priority queue.

From this moment on, q_t is set as the time value attached to the vertical line m, as we know that no point below q_t will be considered anymore.

Outgoing edges of a point q on a vertical line segment are a spiral segment to the next vertical line, and –in case this spiral segment crosses one or more departure curves– segments to vertical lines on other cones. In our implementation, the intersection between spiral segments and departure curves is found using a combination of two approximate root-finding algorithms [3].

Collision-checking straight-line segments is done by testing them for intersections with all cones, except the ones they are tangent to. We approximate a spiral segment between two consecutive vertical lines by one or more small straight-line segments, and collision-check them in the same way (in our implementation, we use a single straight-line segment, as the radial distance ε between two consecutive vertical lines is small).

Finally, the Dijkstra paradigm was replaced by an equally suited A*-method [7], that is faster in practice as it focusses the search to the goal. It adds a lower bound estimate of the distance to the goal to the key-value of each point in the priority queue. In our implementation, the lower bound estimate is simply the Euclidean distance divided by the maximal velocity.

6 Experimental Results

We created an interactive application for planning paths amidst growing discs. The properties of the growing discs (position, size, growth rate) can be changed by the user, and on-the-fly a new path is computed. From this application we report results. Experiments were run on a Pentium IV 3.0GHz with 1 GByte of memory. The value of ε was optimized and fixed at $\frac{2\pi}{40}$.

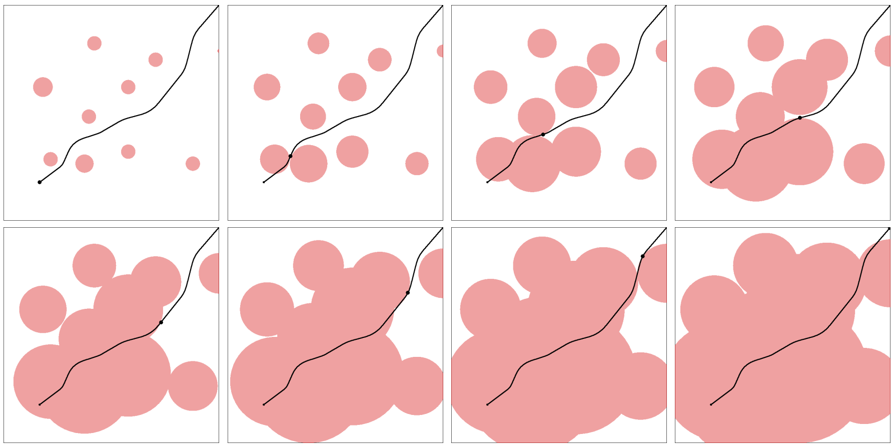

Fig. 6. A shortest path amidst 10 growing discs. A small dot indicates the position along the path at $t = 0, 1, \ldots, 7$. The pictures were generated by our application.

We report the running times of the algorithm for a varying number of discs. As the running time of the algorithm does not only depend on the number of obstacles, but also on the exact configuration of the discs, and how well the A* method manages to focus the search, etc., we averaged the running times over various positions of the start configuration for each experiment. In Fig. 7 the results are given.

What first of all can be seen from the results is that our implementation is very fast. Even for 15 growing discs, the running time is only 0.0042 seconds, well within real-time requirements. We did not show results for more than 15 discs, as it appeared to be difficult to find sensible setups with this many discs that still contain a valid path to the goal. From the figure it seems that the running time is more or less quadratically related to the number of discs. This is what we expected based on the implementation. In Fig. 6, snapshots are shown of a shortest path amidst 10 growing discs.

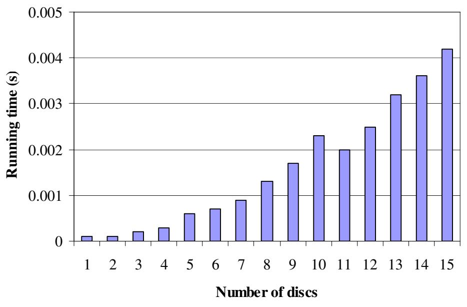

Fig. 7. Results of our experiments

7 Conclusion

In this paper we presented an algorithm for computing shortest paths (minimum time paths) amidst discs that grow over time. A growing disc can model the region that is guaranteed to contain a moving obstacle of which the maximal velocity is given. Hence, using our algorithm, paths can be found that are guaranteed to be collision-free in the future, regardless of the behavior of the moving obstacles. As the regions grow fast over time, a new path should be planned from time to time –based on newly acquired sensor data– to generate paths with more appealing global characteristics. Our implementation shows that such paths can be generated very quickly. A great advantage over other methods is that this replanning can be done safely. The old path that is still used *during* replanning is guaranteed to be collision-free. A requirement though, is that the robot has a higher maximal velocity than any of the moving obstacles.

A drawback of the method we presented is that a path to the goal often does not exist. This occurs when the goal is covered by a growing disc before it can be reached. A solution to this problem would be to find the path that comes closest to the goal. It seems that this can easily be incorporated into our algorithm. Other possible extensions include allowing obstacles with different shapes (other than discs), and fixed obstacles in the environment, but they are still subject of ongoing research.

References

1. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry, Algorithms and Applications, ch. 6 and 8, 2nd edn. Springer, Berlin, Heidelberg (2000)
2. van den Berg, J., Ferguson, D., Kuffner, J.: Anytime path planning and replanning in dynamic environments. In: Proc. IEEE Int. Conf. on Robotics and Automation (ICRA) (2006)
3. Burden, R.L., Faires, J.D.: Numerical analysis, ch.2, 7th edn. Brooks/Cole, Pacific Grove (2001)
4. Chang, E.C., Choi, S.W., Kwon, D.Y., Park, H., Yap, C.K.: Shortest path amidst disc obstacles is computable. In: Proc. Ann. Symposium on Computational Geometry (SoCG), pp. 116–125 (2005)
5. Fiorini, P., Shiller, Z.: Motion planning in dynamic environments using velocity obstacles. Int. J. of Robotics Research 17(7), 760–772 (1998)
6. Hsu, D., Kindel, R., Latombe, J., Rock, S.: Randomized kinodynamic motion planning with moving obstacles. Int. J. of Robotics Research 21(3), 233–255 (2002)
7. LaValle, S.M.: Planning Algorithms, ch. 2. Cambridge University Press, New York (2006)
8. Mitchell, J.S.B.: Geometric shortest paths and network optimization. In: Handbook of Computational Geometry, pp. 633–701. Elsevier Science Publishers, Amsterdam (2000)
9. Petty, S., Fraichard, T.: Safe motion planning in dynamic environments. In: Proc. IEEE Int. Conf. on Intelligent Robots and Systems (IROS), pp. 3726–3731 (2005)
10. Vasquez, D., Large, F., Fraichard, T., Laugier, C.: High-speed autonomous navigation with motion prediction for unknown moving obstacles. In: Proc. IEEE Int. Conf. on Intelligent Robots and Systems (IROS), pp. 82–87 (2004)
11. Weisstein, E.W.: Logarithmic Spiral. In MathWorld – a Wolfram web resource, `http://mathworld.wolfram.com/LogarithmicSpiral.html`