Computing Shortest Paths amidst Growing Discs in the Plane

Jur van den Berg* Mark Overmars*

Abstract

In this paper an algorithm is presented to find a shortest path between two points in the plane amidst growing discs. That is, as the "point" moves through the plane, the discs grow at an a priori known rate. We present an $O(n^3 \log n)$ algorithm and a fast implementation. The problem is motivated from robotics, where motion planning in dynamic environments is a great challenge. Our algorithm can be used to generate paths in such environments guaranteeing that they will be collision-free in the future.

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 dynamic obstacles. In many cases the motions of the dynamic obstacles are not known beforehand, so their future trajectories are estimated by extrapolating current velocities (acquired by sensors) in order to plan a path [4].

A major problem is that the world is continuously changing: If some obstacles change their velocities (say at time t), a new trajectory should be planned. However, there actually is no time for this, no matter how fast it can be done, because at the time the calculation is finished the world has already changed, and hence the computation is outdated. To overcome this problem, often a fixed amount of time, say τ, is reserved for planning. 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 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.

In this paper we describe a technique to overcome these problems. We present an algorithm that computes a path from a start location to a goal location

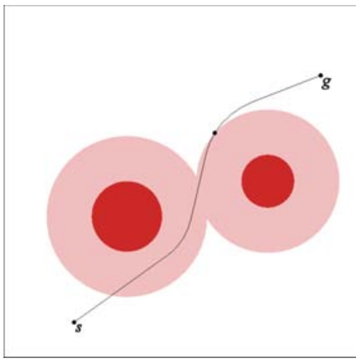

Figure 1: An environment with two dynamic obstacles and a shortest path. The dark and light discs depict the obstacles at $t = 0$ and $t = 1$ respectively. A small dot indicates the position along the path at $t = 1$.

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 this case. The first problem is solved by incorporating all the possible situations of the world at time $t + \tau$ in the world model. As the paths the algorithm computes are guaranteed to be collision-free no matter what the dynamic obstacles do, the second problem dissolves.

We assume that all obstacles and the robot are modeled as discs in the plane, and that the robot and the obstacles have a 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, we model the regions in which the obstacles might be as discs in the plane that grow over time. In this space, we compute a shortest path (a minimum time path) from a start configuration to a goal configuration that is collision-free with respect to the growing discs. Computing shortest paths is a well studied topic in computational geometry (see [5] for a survey). However, the problem posed in this paper presents some interesting challenges of its own. We present an algorithm that runs in $O(n^3 \log n)$ time, where n is the number of obstacles. Further, we created a fast

*Department of Information and Computing Sciences, Universiteit Utrecht, berg@cs.uu.nl, markov@cs.uu.nl

implementation that generates shortest paths at interactive rates.

2 Problem definition

The problem is formally defined as follows. Given are n dynamic obstacles $O_1, \ldots, O_n$ which are discs in the plane. The centers of the discs 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 the same maximal velocity, given by $v \in \mathbb{R}^+$. 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 the maximal velocity of the obstacles, i.e. $V > v$.

As we do not assume any knowledge of the velocities of the dynamic obstacles, other than that they have a maximal velocity, the region that is guaranteed to contain all the dynamic obstacles at some point in time t is bounded by $\bigcup_i B(p_i, r_i + vt)$, where $B(p, r) \subset \mathbb{R}^2$ is an open disc centered at p with radius r. In other words, each of the dynamic 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 $p \notin \bigcup_i B(p_i, r_i + vt)$.*

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

3 Properties

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

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. Observation 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. □

Theorem 2 *A shortest path consists only of straight line segments, and segments of a* logarithmic spiral *incident to the boundary of a growing disc.*

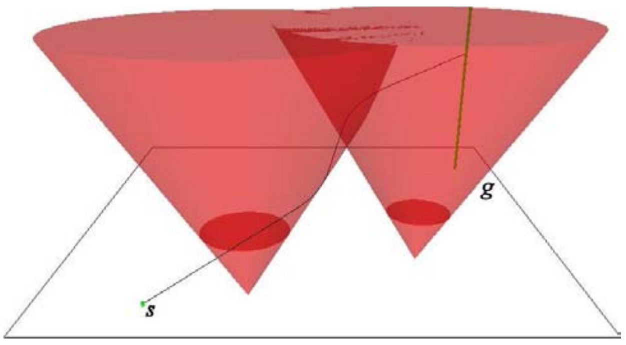

Figure 2: The three-dimensional space of the same environment as Fig. 1.

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 incident to the boundary of a growing disc, it is not possible to shortcut. As both the velocity of the path, and the growth rate of the disc are constant, it is easily shown that such segments are part of a logarithmic spiral [6]. □

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

Proof. Suppose π is a path containing sharp turns. Then these turns could be shortcut by a straight-line segment, and hence π is not a shortest path. Thus, in a shortest path the straight-line segments are tangent to the supporting spirals of the spiral segments. □

4 Global Approach

As the discs grow over time, we can see the obstacles as cones in a three-dimensional space (Fig. 2), where the third dimension models the time. Each obstacle O_i transforms into a cone, 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 determines the opening angle of the cone, and the initial radius r_i determines the (negative) time-coordinate of the apex. The goal configuration 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 this space it is easier to reason about the algorithm we devise to find a shortest path.

Our algorithm to solve the problem is based on a Dijkstra's shortest-path search [3], which starts from the start configuration s. By Theorem 1, all straight-line segments emanating from s of which the slope equals the maximal velocity V, are possible initial motions. This set is narrowed down by Corollary 3, which implies that only the segment leading directly to the goal, and straight-line segments tangent to the cones are possibly part of the shortest path to the goal. These segments may intersect other cones, which would make them invalid, so only the collision-free segments are considered. Each of them is put into a priority queue $\mathcal{Q}$ with a key corresponding to the t-value of its endpoint.

Now, the algorithm proceeds by handling the point with the lowest t-value in the queue (the front element of $\mathcal{Q}$). This point is either the goal location, 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 similarly by finding straight-line segments tangent to other cones and to the goal configuration. However, as we are on the surface of a cone, we first have to walk a piece of a spiral around the cone such that the straight-line segment is tangent to both the 'source' cone and the 'destination' cone. For each cone, as well as for the goal configuration, these segments are computed and if collision-free their endpoints are inserted in $\mathcal{Q}$.

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, no valid path exists.

5 Details

The algorithm described above will indeed find a shortest path to the goal. However, in order to have a finite 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.

Let us look at the following. A shortest path consists of spiral segments on a cone's surface and straight-line segments that are bitangent to two cones. There are four ways in which a segment can be bitangent to a pair of cones: left-left, right-right, left-right and right-left. In each of these cases, there is an infinite number of bitangent segments with a slope corresponding to the maximal velocity V, but the possible tangency points at the source cone form a continuous curve on the surface of the cone. We call such curve a *departure curve*.

The departure curve may be cut into several collision-free *intervals* by other cones that penetrate the surface of the cone. These intervals form the 'nodes' in our search process. Only the path arriving earliest in an interval can contribute to a shortest path. Other 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. This argument applies if for the departure

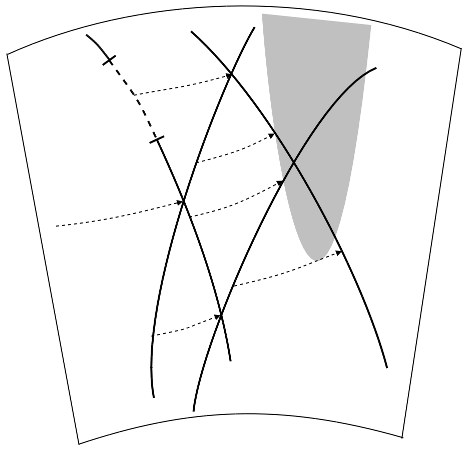

Figure 3: An impression of an arrangement on the surface of the cone. The thick lines are the departure curves, of which one has a shadow interval (dashed). The thin dashed lines are spiral segments that delimit trapezoidal regions that have the same next departure curve or collision (only the counter-clockwise spirals are shown). The gray area depicts an intersection area of another cone penetrating the surface, and cutting several departure curves into two intervals.

curve holds that $\frac{||(\delta x, \delta y)||}{\delta t} \leq V$. This is only the case when all cones have the same opening angle. (This explains our assumption that all discs have the same maximal velocity v.) As the proof is rather technical, we omit it here.

To identify these departure curve intervals, we compute an *arrangement* [1] on the surface of each cone of all departure curves on that cone (see Fig. 3). Areas on the surface on the cone that are intersected by other cones are also inserted in this arrangement. The intervals can then be extracted from the arrangement.

Each departure curve interval has two outgoing edges. One –a spiral segment– to the next departure curve on the source cone, and one –consisting of a bitangent straight-line segment and a spiral segment– to the first departure curve encountered on the destination cone that is associated with the departure curve. 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 around the cone. This can be done 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 crosses to another cone, we have to determine what the first departure curve is we will encounter there. This is done 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 spiral's cone surface. Since intersection 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. 3, an impression is given of how such an arrangement might look.

Theorem 4 *The algorithm to compute a shortest path amidst n growing discs 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 $O(n)$ cones possibly intersecting the departure curve. (Each cone can split the departure curve in 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. $\square$

6 Implementation

We created a fast implementation of the algorithm presented above. Instead of using the arrangements, we used some ad-hoc approaches for doing the elementary tests. These may have a slower asymptotic running time, but in practice they turned out to be fast. For example, intersections between a spiral and a departure curve cannot be found analytically, so we used a combination of two approximate root-finding algorithms [2]. The Dijkstra method was replaced by an equally suited A*-method [4], that is faster in practice as it focusses the search to the goal. The implementation runs at interactive rates even for many obstacles. For example, a shortest path among 10 cones is computed within 0.01 seconds on a Pentium IV 3.0GHz with 1 GByte of memory. Figs. 1 and 2 were created using our implementation.

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 could model the region that is guaranteed to contain a dynamic 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 dynamic 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 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 dynamic 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 in our algorithm, but it is still subject of ongoing research.

References

[1] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf. Computational Geometry, Algorithms and Applications. Chapters 6 and 8. Springer-Verlag, Berlin Heidelberg, 1997.

[2] R. L. Burden, J. D. Faires. Numerical analysis, 7th edition. Chapter 2. Brooks/Cole, Pacific Grove, 2001.

[3] E. W. Dijkstra. A note on two problems in connexion with graphs. Numerische Mathematik, 1:269-271, 1959.

[4] S. M. LaValle. Planning Algorithms. Cambridge University Press, 2006.

[5] J. S. B. Mitchell. Geometric shortest paths and network optimization. In Handbook of Computational Geometry, pages 633-701. Elsevier Science Publishers, Amsterdam, 2000.

[6] E. W. Weisstein. Logarithmic Spiral. In MathWorld – A Wolfram Web Resource. http://mathworld.wolfram.com/LogarithmicSpiral.html