Jur van den Berg
Department of Computer Science,
University of North Carolina at Chapel Hill, Chapel Hill, NC, USA
berg@cs.unc.edu

Mark Overmars
Department of Information and Computing Sciences,
Utrecht University, Utrecht, The Netherlands
markov@cs.uu.nl

Planning Time-Minimal Safe Paths Amidst Unpredictably Moving Obstacles

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 maximum speeds of the moving obstacles, which we assume are disk-shaped, the regions that are possibly not collision-free are disks that grow over time. We present an approach to compute the time-minimal path *between two points in the plane that avoids these growing disks. The generated paths are thus guaranteed to be collision-free with respect to the moving obstacles while being executed. We present an algorithm that runs in $O(n^3 \log n)$ time (n being the number of obstacles) for the case where the moving obstacles have the same maximum speed, and a fast implementation for the general case that is capable of planning paths amidst many growing disks within milliseconds.*

KEY WORDS—Motion planning, moving obstacles, shortest paths

1. Introduction

An important challenge in robotics is real-time online motion planning in dynamic environments. That is, navigating a robot from a start location to a goal location while avoiding collisions with the moving obstacles. In many cases the motions of the moving obstacles are not known beforehand, so in order to plan a path, the planner often relies on estimates of their future trajectories, which are obtained, for instance, by extrapolating the current velocities of the moving obstacles (Fiorini and Shiller 1998; Vasquez et al. 2004; van den Berg et al. 2006; Zucker et al. 2007). As the world is continuously changing, these estimates may only prove to be valid for a short amount of time. Therefore, most online planners employ a continuous cycle of sensing and (re)planning during the motion of the robot, in order to quickly react to changes in the environment (Petty and Fraichard 2005; Vannoy and Xiao 2006).

An important issue in online planning, that has received a lot of attention over recent years, is safety (Frazzoli et al. 2002; Fraichard and Asama 2004; Petty and Fraichard 2005; Bekris and Kavraki 2007). As planning takes time, there is an inherent latency between observing and acting, which may jeopardize the safety of the robot. Let us look at the planning cycle in some more detail. Let the period of the cycle be τ, and let t be the time at the beginning of a cycle. Then, the planner has until time $t + \tau$ to plan a path, for which it uses obstacle trajectory estimates that are based on information acquired at time t. After planning has finished, the path is used to guide the robot between time $t + \tau$ and time $t + 2\tau$. The next cycle (starting at time $t + \tau$) plans a path that is used between time $t + 2\tau$ and time $t + 3\tau$, and so on.

This means that, in order to guarantee immediate safety for the robot, the estimates of the obstacle trajectories obtained at the beginning of each cycle should be valid for at least 2τ time. Guaranteeing this safety is a difficult task, even more so when τ is large. Choosing a small τ is often not an option, as this may leave too little time for planning a path with enough look-ahead to bias the robot towards the goal, and guarantee safety in the long run (for instance, to prevent the robot from ending up in an inevitable collision state (Fraichard and Asama 2004)).

In this paper we take a first step towards overcoming this problem. We present an approach to compute the time-minimal 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 based on new obser-

The International Journal of Robotics Research
Vol. 27, No. 11–12, November/December 2008, pp. 1274–1294
DOI: 10.1177/0278364908097581

Figures 1–5, 7–11 appear in color online: http://ijr.sagepub.com

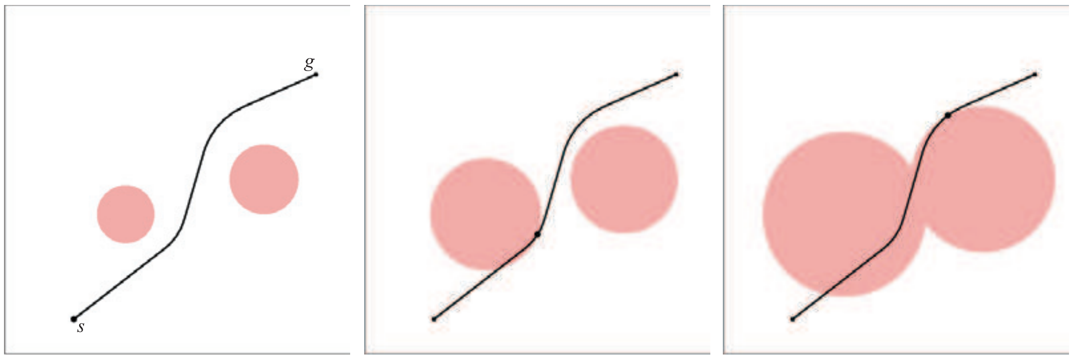

Fig. 1. An environment with two moving obstacles and a time-minimal path. The growing disks are shown at $t = 0$, $t = 1$ and $t = 2$, respectively. A small dot indicates the position along the path.

vations might still be necessary from time to time, to generate paths with more appealing global characteristics, but our planner allows a conveniently large value to be chosen for τ, 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 disks in the plane, and that the robot and each of the obstacles have a (known) maximum speed. The maximum speed of the obstacles should not exceed the maximum speed 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.

The key observation we use in this paper is that given the initial positions and maximum speeds of each of the obstacles, the regions of the space that are possibly not collision-free are disks that grow over time with rates corresponding to the maximum speeds of the obstacles. Our goal is to compute a *time-minimal path* from a start to a goal position that avoids these growing disks (see Figure 1).

Although computing time-minimal and shortest paths is a well-studied topic in computational geometry (see Mitchell (2000) for a survey), the problem we study in this paper is new. In fact, it is a three-dimensional shortest path problem lying in the *configuration-time space* (Latombe 1991), 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 disks) for our problem in the restricted case that all disks have the same growth rate[1]. In case the growth rates are different, we cannot give a time bound expressed in n. Instead, we implemented a practical heuristic algorithm for this general case, which appears to work well: experimental results show that we are able to generate time-minimal paths amidst many growing disks within only milliseconds of computation time.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of related work. We formally define the problem in Section 3. In Section 4 we examine the structure of time-minimal paths amidst growing disks, and in Section 5 we define some notions that are important when constructing our algorithm. We sketch our global approach in Section 6, and in Section 7 we present an efficient algorithm for the restricted case and prove a running time bound of $O(n^3 \log n)$. In Section 8, we consider the general case, and present an efficient implementation of a heuristic algorithm, as well as experimental results. Section 9 discusses future work and concludes the paper.

2. Related Work

Path planning in dynamic environments has been an active area of research in robotics. The methods that have been proposed can roughly be classified into two categories. The first category of methods plan a path in a *static* configuration space, and *replan* when it appears that an obstacle has moved (Stentz 1995; Leven and Hutchinson 2002; Jaillet and Simeon 2004; Kallman and Mataric 2004; Vannoy and Xiao 2006). These approaches are highly effective in environments where the exact location of obstacles is not precisely known, or in *changing* environments where obstacles are not expected to move very fast or very often. They are less suitable in environments with high-speed moving obstacles (for instance, when crossing a busy highway), however, as the methods do not specifically take into account any information that may be available regarding the future trajectory of the obstacles.

Methods that do take into account the future trajectories of obstacles plan paths in the *configuration-time space*, in which time accounts for an additional dimension (Hsu et al. 2002;

1. Note that the special case of disks 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., Chang et al. (2005)).

Vasquez et al. 2004; van den Berg et al. 2006; Zucker et al. 2007). Planning in configuration-time spaces allows for temporally coordinating the motion of the robot with the motions of the obstacles. Often, the future motions of the obstacles are not precisely known, so the planners rely on estimates of the future trajectories, which are obtained, for instance, by extrapolating current velocities (Fiorini and Shiller 1998). Owing to the inherent imprecision in these estimates, the planned motions may not be 100% safe. In this paper, we present a method that uses the worst-case estimate of the future trajectories of the obstacles, and is hence guaranteed to be safe.

We formulated the problem we discuss in this paper as finding a time-minimal path among growing disks in the plane. The concepts we use in our paper are somewhat reminiscent of those used by Fujimura and Samet (1993), in which an $O(n^2 \log n)$ algorithm is presented that finds a time-minimal path among moving polygonal obstacles with known linear trajectories.

3. Problem Definition

The problem we discuss in this paper is formally defined as follows. There are n moving obstacles $O_1, \ldots, O_n$ which are disks in the plane. The centers of the disks (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 disks by $r_1, \ldots, r_n \in \mathbb{R}^+$. All of the obstacles have a maximum speed, given by $v_1, \ldots, v_n \in \mathbb{R}^+$. The robot is a point (if it is a disk, 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 position $s \in \mathbb{R}^2$ and a goal position $g \in \mathbb{R}^2$. The robot has a maximum speed $V \in \mathbb{R}^+$ which should be larger than each of the maximum speeds 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 maximum speed, the region that is guaranteed to contain all of 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 disk centered at p with radius r. In other words, each of the moving obstacles is conservatively modeled by a disk that grows over time with a rate corresponding to its maximum speed (see Figure 1 for an example environment).

We define a point to be collision-free at some time t if it is not contained in any of the growing disks at time t.

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

We can now formally define the problem we solve in this paper: the goal is to compute a path $\pi : [0, t_g] \to \mathbb{R}^2$, such that:

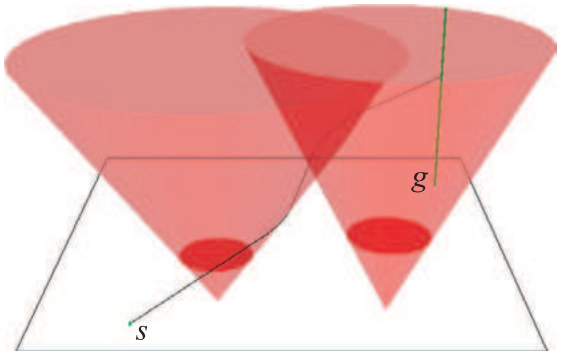

Fig. 2. The three-dimensional configuration-time space of the same environment as shown in Figure 1.

- $\pi(0) = s$ and $\pi(t_g) = g$;
- $\forall t \in [0, t_g] :: \pi(t)$ is collision-free at time t;
- $\forall t_1, t_2 \in [0, t_g] : t_1 < t_2 : \|\pi(t_2) - \pi(t_1)\| / (t_2 - t_1) \leq V$; and
- π is a *time-minimal path*, i.e. t_g is minimal among all paths obeying the above conditions.

4. Properties of Time-Minimal Paths

In this section we deduce some elementary properties of time-minimal paths amidst growing disks, which we later use to devise an efficient algorithm to compute time-minimal paths. We first show that we are actually dealing with a three-dimensional path planning problem: as the disks grow over time, we can see the obstacles as cones in a three-dimensional configuration-time space (see Figure 2), where the third dimension represents the time (Latombe 1991). 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 maximum speed 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

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

The goal position 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 time-minimal paths.

4.1. Maximum Speed

The first property we prove is that a time-minimal path is always traversed at the maximum speed V, and hence a time-minimal path makes a constant angle $\arctan(1/V)$ with the xy-plane in the three-dimensional configuration-time space.

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

Proof. If $t_1 \le t_2$, 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, that is, $p \notin \bigcup_i B(p_i, r_i + v_i t_2)$, 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 3. *The speed $\|(d\pi_x/dt, d\pi_y/dt)\|$ of a time-minimal path π is constant and equal to the maximum speed V.*

Proof. Suppose that π is a path to g, of which a sub-path has a speed smaller than V. Then this sub-path could have been traversed at maximum speed, so that points further along the path would be reached at an earlier time. Lemma 2 proves that these points are then collision-free as well, so g could also have been reached sooner, and hence π is not a time-minimal path. □

The *length* $L(\pi)$ of a path π that arrives at the goal g at time t_g is defined as

$$L(\pi) = \int_0^{t_g} \left\| \left(\frac{d\pi_x}{dt}, \frac{d\pi_y}{dt} \right) \right\| dt.$$

By Theorem 3, the length of a time-minimal path π equals $L(\pi) = V t_g$.

Corollary 4. *A time-minimal path π is the shortest (in terms of length) among all valid collision-free paths between s and g.*

Proof. Let π be a time-minimal path arriving at the goal g at time t_g. As stated above, the length of π is $L(\pi) = V t_g$. Suppose that there is a shorter path π' that has length $L' < V t_g$. Then, following similar arguments as made in the proof of Theorem 3, this path can be traversed with speed V and reaches the goal g in $L'/V < t_g$ time. Hence, π is not a time-minimal path. □

4.2. Straight-line Segments and Spiral Segments

The next property we prove is that a time-minimal path can only consist of straight-line motions, and motions that stay in contact with the growing disks. These latter motions follow spiral curves "winding" around a growing disk. They lie on the surface of a cone, when viewed in the three-dimensional space. In fact, as both the speed of the path and the growth rate of the disks are constant, the motions on the boundary of a disk are supported by a *logarithmic spiral*.

Lemma 5. *Maximum speed motions on the boundary of a growing disk follow a curve supported by a logarithmic spiral.*

Proof. Without loss of generality, we assume that the disk has radius 0 at $t = 0$, that the disk is centered at the origin, and that the disk grows with speed 1 (other configurations can be transformed such that these conditions hold). Let the speed of the path be V. We express the equations of the path curve in polar coordinates $(r(t), \theta(t))$, parameterized by the time t.

The radius $r(t)$ of the curve at time t is equal to the radius of the disk at time t, so, as the disk grows with speed 1:

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

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

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

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

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

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

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

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

Theorem 6. *A time-minimal path π solely consists of straight-line segments, and logarithmic spiral segments on the boundary of a growing disk.*

2. See http://mathworld.wolfram.com/LogarithmicSpiral.html.

Proof. Any point along π *either* lies on the boundary of a growing disk, *or* in open space (meaning that there is a ball of positive radius around the point that does not intersect any obstacle). In Lemma 5 we have seen that in the first case this point lies on a curve supported by a logarithmic spiral. In the second case, it is trivial that the point lies on a straight-line segment: Theorem 3 implies that the time it takes to traverse a path is proportional to its length. Therefore, 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 disk it is not possible to shortcut. Hence, a time-minimal path π solely consists of straight-line segments, and logarithmic spiral segments. □

4.3. Path Smoothness

Lastly, we prove that a time-minimal path contains no sharp turns, i.e. the path is C^1 or continuously differentiable.

Theorem 7. *A time-minimal path π contains no sharp turns.*

Proof. Above we have seen that a time-minimal path consists of straight-line segments and spiral segments only. As these segments do not contain sharp turns themselves, a sharp turn possibly only occurs where two segments are connected. Suppose that a path π contains such sharp turns. Let $\pi(t^*)$ be a point along path π where a sharp turn occurs. Then, there exist a $\delta > 0$ such that the points $\pi(t^* - \delta)$ and $\pi(t^* + \delta)$ can be connected by a straight-line segment that is collision-free. Obviously, the straight-line shortcut segment is *shorter* (in terms of length) than the segment of path π between $\pi(t^* - \delta)$ and $\pi(t^* + \delta)$. This contradicts Corollary 4, and therefore π is not a time-minimal path.

We can prove as follows that such a collision-free shortcut exists. Let us consider the projection of the path π onto the plane. We only have to make sure that the shortcut is collision-free with respect to growing disks whose center points lie on the same side of path π as the shortcut segment. Growing disks on the other side of the path cannot invalidate the shortcut, as otherwise the original path π would not be collision-free.

Let us consider a growing disk O_i with growth rate v_i that might invalidate the shortcut. Obviously, the sharp turn point $\pi(t^*)$ does not lie inside O_i or on the boundary of O_i at time t^*, as otherwise either the portion of path π just before time t^*, or the portion of π just after time t^* would not be collision-free. Hence, the (Euclidean) distance d_i^* between point $\pi(t^*)$ and O_i at time t^* is positive, i.e. $d_i^* > 0$.

Let $\tilde{\pi}$ be the path π with the portion between $\pi(t^* - \delta)$ and $\pi(t^* + \delta)$ replaced by the straight-line shortcut segment. The distance between point $\pi(t^*)$ and any point $\tilde{\pi}(t)$ (with $t \in [t^* - \delta, t^* + \delta]$) along the shortcut is smaller than δV (where V is the speed along path π). Let $\tilde{d}_i(t)$ be the distance between point $\tilde{\pi}(t)$ along the shortcut and growing disk O_i at time t. We know the following about $\tilde{d}_i(t)$:

$$\forall t \in [t^* - \delta, t^* + \delta] :: \tilde{d}_i(t) \geq d_i^* - \delta V - (t - t^*)v_i$$

$$\geq d_i^* - \delta V - \delta v_i = d_i^* - \delta(V + v_i).$$

Hence, a shortcut with $\delta < d_i^*/(V + v_i)$ is collision-free with respect to growing disk O_i, and a shortcut with $\delta < \min_i(d_i^*/(V + v_i))$ is collision-free with respect to all growing disks.

Thus, a time-minimal path does not contain any sharp turns. □

This theorem implies that in a (general) time-minimal path the straight-line segments and spiral segments alternate, and that the straight-line segments must be *tangent* to the supporting spirals of the spiral segments[3]. The first and the last segment of the path are straight-line segments connected to the start and the goal position, respectively, and all other straight-line segments are, in terms of the three-dimensional space, *bitangent* to the cones on which the spirals lie.

5. Definitions

Based on the properties of time-minimal paths deduced in the previous section, we introduce a number of notions in this section that play an important role in our algorithm to compute a time-minimal path. These are: *departure curves* (Section 5.1), *obstacle regions* (Section 5.2), *wedge regions* (Section 5.3), and *departure curve intervals* (Section 5.4). All of these notions "live" on the surfaces of the cones.

5.1. Departure Curves

As established in the previous section, the straight-line segments of a time-minimal path are *bitangent* to the cones on which the spiral segments of the path lie. There are four ways in which a (directed) straight-line segment can be bitangent to a pair of cones (say C_i and C_j): left–left, left–right, right–right, and right–left (see Figure 3)[4]. In each of these cases, there is an infinite number of possible segments (whose slope corresponds to the maximum speed 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 time-minimal path. Similarly, arrival curves can be defined on the surface of C_j.

3. Two spiral segments can succeed each other only in the degenerate case at the moment when two growing disks collide. In this case, we say that the two spiral segments are connected by an infinitesimally short straight-line segment.

4. Left and right refer to the side of the (center of the) cone on which the directed segment lies when projected onto the xy-plane.

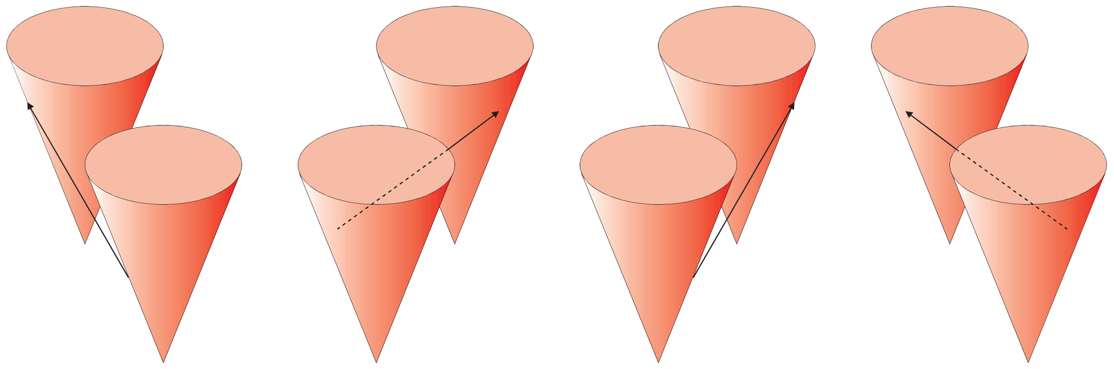

Fig. 3. The four bitangency cases of a (directed) straight-line segment and a pair of cones: (a) left–left; (b) left–right; (c) right–right; (d) right–left.

Definition 8. (departure curve). *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 by $DC_{ll}(C_i, C_j)$, $DC_{lr}(C_i, C_j)$, $DC_{rl}(C_i, C_j)$, and $DC_{rr}(C_i, C_j)$, respectively.*

The set $DC(C_i, g)$ to the goal position g is defined similar, but then the tangent line segment should go through the goal position 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 disk associated with the cone has radius 0 at $t = 0$, that the disk is centered at the origin, and that the disk grows with speed 1. Let the speed of the path be V. The surface of C can be parameterized by two variables, angle Θ and time T:

$$C : (\Theta, T) \to (T \cos\Theta, T \sin\Theta, T). \tag{6}$$

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 (Θ, T) 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:

$$\theta_0 = -\sqrt{V^2 - 1}\log T + \Theta. \tag{7}$$

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

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

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

$$\begin{aligned} \ell(t) &= \left\{x(T) + (t-T)x'(T),\ y(T) + (t-T)y'(T)\right\} \\ &= \left\{t\cos\Theta - (t-T)\sqrt{V^2-1}\sin\Theta,\right. \\ &\qquad \left. t\sin\Theta + (t-T)\sqrt{V^2-1}\cos\Theta\right\}. \end{aligned} \tag{8}$$

This segment must be tangent to cone C_i (that has position p_i, initial radius r_i, and speed v_i), in order for point (Θ, T) 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 Equation (1), by substituting $x = \ell_x(t)$ and $y = \ell_y(t)$, we obtain a quadratic equation in variable t for the intersection points of line ℓ and cone C_i. Solving for t gives solutions of the familiar form:

$$t = \frac{a(\Theta, T) \pm \sqrt{D(\Theta, T)}}{V^2 - v_i^2}. \tag{9}$$

The expressions for $a(\Theta, T)$ and $D(\Theta, T)$ are given in Appendix A.

Here, $D(\Theta, T)$ is the *discriminant* whose sign indicates how many solutions the above equation has (0, 1, or 2). If $D(\Theta, T) > 0$, there are two solutions meaning that ℓ intersects C_i. If $D(\Theta, T) < 0$, there are no solutions meaning that ℓ does not intersect C_i. If $D(\Theta, T) = 0$, there is one solution, meaning that ℓ is *tangent* to C_i. Hence, $D(\Theta, T) = 0$ is an implicit equation for the set $DC_{rr}(C, C_i) \cup DC_{rl}(C, C_i)$. We can make this explicit by solving $D(\Theta, T) = 0$ for T. In Figure 4 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.

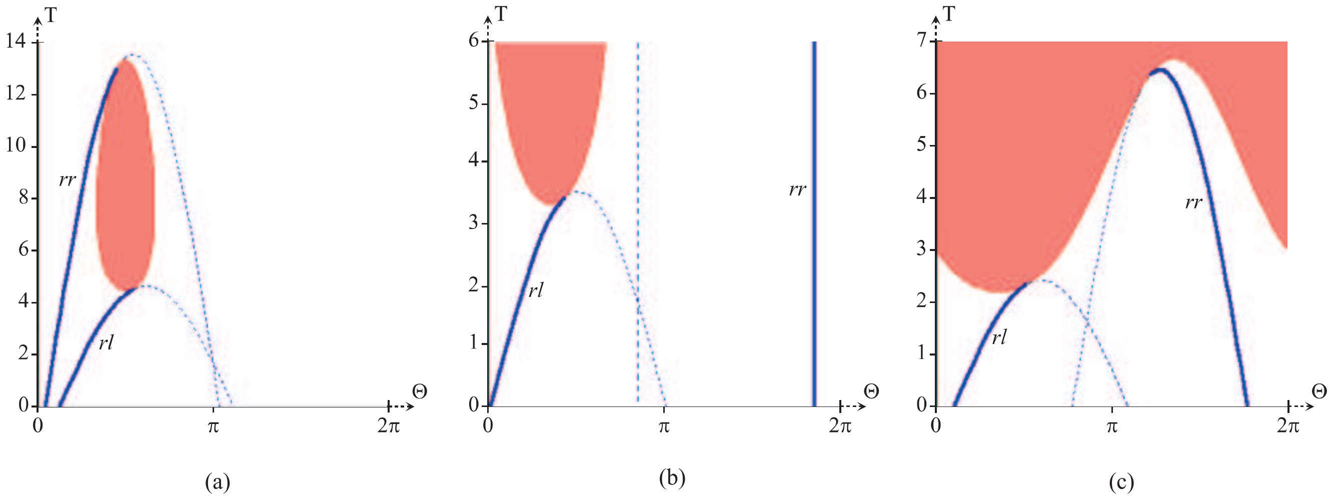

Fig. 4. Departure curves $DC_{rr}(C, C_i)$ and $DC_{rl}(C, C_i)$ on the surface of C parameterized by angle Θ and time T for different values of v_i (i.e. the growth rate of C_i). (a) If $v_i < 1$, the growth rate of C_i is smaller than the growth rate of C. (b) If $v_i = 1$, the growth rates are equal, and (c) if $v_i > 1$, the growth rate of C_i is larger than the growth rate of C. The dashed curves are arrival curves. The gray area is an obstacle region, i.e. the region on the surface of C that is penetrated by C_i. These points are not collision-free.

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 (Θ, T) on the surface of cone C for which $D(\Theta, T) = 0$, the arrival time of the straight-line segment at cone C_i is given by $A(\Theta, T) = a(\Theta, T)/(V^2 - v_i^2)$. The departure time of the segment is given by T. Along some parts of the departure curve, $A(\Theta, T)$ is smaller than T, that is, the arrival time on C_i is smaller than the departure time at C. These parts actually correspond to arrival curves of segments departing at C_i and arriving at C. In the plots this is indicated by dashed curves. In the remainder of this paper these arrival curves are ignored when we refer to departure curves.

We also have to take into account the departure curves of $DC(C, g)$ associated with segments tangent to C and leading to the goal position 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.

5.2. Obstacle Regions

Another important notion in our algorithm is the *obstacle regions* on the surface of a cone. They are induced by other cones that penetrate the surface. These parts of the surface of the cone are inaccessible, and cannot be traversed by a spiral around the cone.

Definition 9. (Obstacle region) *An* obstacle region *on the surface of a cone C induced by another cone C_i is defined as the set of points on the surface of C that are inside cone C_i penetrating the surface of C. Hence, the points in the obstacle region are not collision-free.*

We now show how we can deduce equations for the obstacle regions on the surface of a cone C. Again, without loss of generality, we assume that the disk associated with C has radius 0 at $t = 0$, that the disk is centered at the origin, and that the disk grows with speed 1.

A point (Θ, T) on C is inside another cone C_i with position p_i, initial radius r_i, and speed v_i, if

$$(T\cos\Theta - p_{ix})^2 + (T\sin\Theta - p_{iy})^2 < (r_i + v_i T)^2. \quad (10)$$

This is simply obtained by combining Equations (6) and (1). Hence, the obstacle region on the surface of C induced by cone C_i is given by the above equation. In Figure 4, we see obstacle regions for various values of v_i.

Changing the $<$ to an $=$ in Equation (10) and solving for T, gives a closed form for the boundary of the obstacle region.

5.3. Wedge Regions

If there exists a collision-free path with speed $\leq V$ from the start position s to a point (Θ, T) on the surface of a cone, all points on the cone that are reachable from (Θ, T) by following some collision-free path on the surface of the cone with a speed less than the maximum speed V can never lie

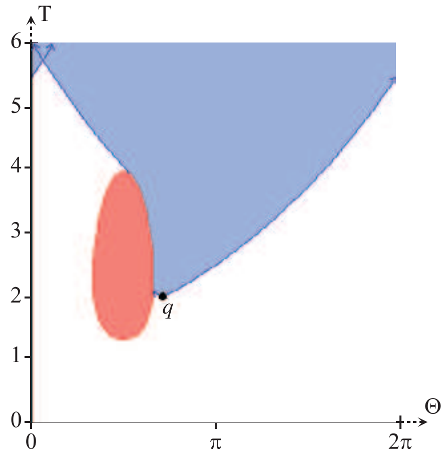

Fig. 5. The wedge region (light gray) of a point $q = (\Theta, T)$ contains the points on the surface of the cone that are reachable from point $q = (\Theta, T)$ by paths on the surface of the cone with a speed less than or equal to V. The dark gray area is an obstacle region.

on a time-minimal path from the start to the goal (this follows directly from Theorem 3). These points are contained in the wedge formed by the clockwise and counterclockwise spiral going through (Θ, T), as we know that on the spirals the speed equals V (see Figure 5; a spiral appears as an exponential function in the ΘT-coordinate frame). We call this region the *wedge region* of (Θ, T).

Definition 10. (Wedge region). *A* wedge region *of a point* (Θ, T) *on the surface of a cone C is defined as the set of points on the surface of C to which a collision-free path from* (Θ, T) *exists that lies on the surface of C, and has speed at most V at any point along the path.*

5.4. Departure Curve Intervals

A departure curve on the surface of a cone may be subdivided into a number of collision-free segments by the obstacle regions on the surface of that cone. We call those collision-free segments *departure curve intervals*.

Definition 11. (Departure curve interval). *A* departure curve interval *is a maximal continuous* collision-free *segment of a departure curve.*

In the specific case in which all disks have the same growth rate, say $v_i = v < V$ for all i (see Figure 4(b)), these departure curve intervals satisfy an interesting property: let (Θ, T) be a point on some departure curve interval, then all points (Θ', T') on the same departure curve interval for which $T' > T$ are within the wedge region of (Θ, T). To prove this, we must show that a path following a departure curve on the surface of a cone has a speed less than or equal to V.

Lemma 12. *In case the growth rate of all disks are equal, a path following a departure curve on the surface of a cone has a speed less than or equal to V.*

Proof. See Appendix B. □

We use this property in Section 7 to devise an efficient algorithm to find a time-minimal path in case the growth rates of all disks are equal.

6. Global Approach

With the notions introduced so far, we can devise a first, rather naive algorithm to find a time-minimal path amidst growing disks from some start position s to some goal position g. It sketches the global approach we take and it serves as the basis for the efficient algorithms we present in Sections 7 and 8.

6.1. The Basic Algorithm

Our approach grows a tree of possible time-minimal paths that is rooted in the start position at time $t = 0$. A leaf is *expanded* if the length of its path from the start position is minimal among all leaves of the tree. To this end, each leaf is maintained in a *priority queue*, with a *key value* equal to its time coordinate (which is proportional to the length of its path from s). The priority queue is initialized with the initial motions from the start position s that possibly belong to a time-minimal path. These are straight-line segments with slope $1/V$ leading either directly to the goal position, 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 position, in which case the time-minimal 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 region, 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 (or

Algorithm 1 TIMEMINIMALPATHBASIC(s, g)

```
1: for all endpoints $(q, t)$ of the collision-free segments from $s$ that are tangent to a cone do
2:     Insert $(q, t)$ into priority queue $Q$.
3: while $Q$ is not empty do
4:     Pop element $(q, t)$ with minimal $t$ from the queue.
5:     if the goal position is not collision-free anymore at time $t$ then
6:         Path does not exist. Terminate.
7:     else if $q = g$ then
8:         Time-minimal path found! Terminate.
9:     else
10:        $(q, t)$ is on the surface of a cone, say $C_i$, so proceed along the spiral about $C_i$ until it runs into an obstacle region, or encounters a departure curve.
11:        if the spiral encounters a departure curve, say $DC(C_i, C_j)$, then
12:            $(q', t') \leftarrow$ the intersection point of the spiral and the departure curve.
13:            $(q'', t'') \leftarrow$ arrival point of the bitangent segment on the surface of $C_j$.
14:            Insert $(q', t')$ into $Q$.
15:            if straight-line segment from $(q', t')$ to $(q'', t'')$ is collision-free then
16:                Insert $(q'', t'')$ into $Q$.
17: Path does not exist.
```

the goal position) by a straight-line segment. If this latter segment is collision-free, its endpoint is inserted into the queue. An entry is also enqueued for the first option.

This procedure is repeated until the goal position is popped from the priority queue. In this case the time-minimal 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 position is no longer collision-free (it is occupied by one of the growing disks), no valid path exists. The algorithm is given in pseudocode in Algorithm 1.

6.2. Identifying Spirals

In Algorithm 1, we have to identify the spiral we are on (let us assume that it is a counterclockwise spiral), given the point on the surface of the cone where we arrived (line 9). Let (q, t) be a point on the surface of some cone, say C_i, where q is given in Euclidean coordinates (x, y). Then the corresponding coordinates (Θ, T) on the surface of C_i are given by[5] $(\Theta, T) = (\arctan(q_y - p_{iy}, q_x - p_{ix}), t)$.

5. We use the notation $\arctan(y, x)$ here to refer to the quadrant-aware version of the inverse tangent.

The spiral on the surface of C_i going through (Θ, T) is given by θ_0 as computed in Equation (7). Equation (5) then gives a function for the angle $\theta(t)$ along the spiral through (Θ, T). 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 $D(\Theta, T) = 0$ for the departure curves of any pair of cones (see Equation (9)). The intersections are thus found by solving $D(\theta(t), t) = 0$ for t. Note that when we are on a counterclockwise spiral, we are only interested in rl- and rr-departure curves. If we are on a clockwise spiral, we wish to find intersections with lr- and ll-departure curves.

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 $A(\theta(T), T)$, as defined in Section 5.1. If this arrival time is smaller than T, the intersection can be ignored. If it is larger, we wish to know whether the tangent straight-line segment arrives on the left-hand side of C_j (and should be succeeded by a clockwise spiral on C_j), or on the right-hand side of C_j (and should be succeeded by a counterclockwise spiral). This is determined by the derivative of $D(\theta(t), t)$ to t. If this derivative is negative at point T, we have arrived on the left-hand side. If it is positive, we have arrived on the right-hand side[6]. The exact arrival location on the surface of cone C_j is given by $\ell(A(\theta(T), T))$ (see Equation (8)). From this information we can deduce the parameters defining the spiral on C_j on which we have arrived.

7. The Restricted Case: Disks with Equal Growth Rates

The algorithm described above will indeed find a time-minimal 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 time-minimal path, such that we can perform *relaxation* on them as in Dijkstra's algorithm (LaValle 2006). In this section, we show that it is possible to achieve this in the restricted case where all disks have equal growth rates, and present an $O(n^3 \log n)$ algorithm for computing the time-minimal path (n being the number of disks). For the general case this problem

6. This can be seen as follows: as we walk along the *counterclockwise* spiral on C_i, the line ℓ (see Equation (8)) tangent to the spiral "radially sweeps" the space counterclockwisely. If the tangent line "enters" a cone C_j (which happens necessarily on the right-hand side of C_j), the discriminant $D(\theta(t), t)$ is negative (no intersections) just before entering the cone, and positive just after entering the cone (two intersections). Hence, its derivative is positive at the point where C_j is hit. The opposite holds when the tangent line ℓ "leaves" C_j (which happens necessarily on the left-hand side of C_j).

is left open, but we present a heuristic algorithm in Section 8 that is very fast in practice, as it prunes large parts of the search tree.

7.1. An Efficient Algorithm

To devise an efficient algorithm, we use the fact following from Lemma 12 that the *departure curve intervals* (see Section 5.4) can only be visited once by a time-minimal path. Only the path arriving earliest in a departure curve interval can contribute to a time-minimal path: paths arriving later in the interval cannot be part of the time-minimal path, because the path arriving earliest in the interval can be extended with a collision-free traversal at speed at most V along the departure curve interval to end up at the same position (and time) as the path arriving later in the interval. This means that these departure curve intervals can serve as *nodes* in our Dijkstra algorithm.

Each node (a departure curve 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 encountered 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.

The efficient algorithm follows the same approach as the basic algorithm presented in the previous section. However, we now stop exploring branches in the time-minimal path tree if they arrive in a departure curve interval that has already been visited at an earlier time by another branch in the tree. To this end, we maintain for each departure curve interval a time-value at which it was last visited. Initially, these are set to ∞. If a path arrives at a departure curve interval at a time earlier than the maintained time-value, we update the interval's time-value, and insert the departure curve interval into the priority queue. If a path arrives at an interval at a time later than the stored time-value, we do not expand this path any further. The pseudocode of the efficient algorithm is given in Algorithm 2 (in the pseudocode, we treat the goal as a special "departure curve interval").

For each edge that we handle during the algorithm, we have to do three things: first, if the edge is a traversal to another cone, we have to identify whether we arrived on the left-hand side of the cone (and should continue on a clockwise spiral) or on the right-hand side of the cone (and should continue on a counterclockwise spiral); see lines 6 and 33 in Algorithm 2. Second, we have to find the departure curve interval (or obstacle region) in which it will arrive (lines 6, 23, and 33), and third, we must determine whether the edge is collision-free (lines 5 and 32). The first operation has been described in Section 6.2. In the next section, we present a datastructure which allows the latter two operations to be performed efficiently.

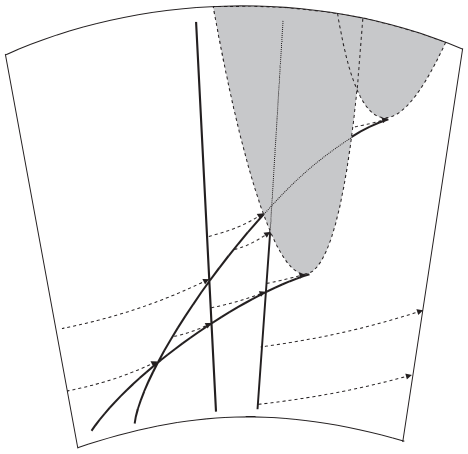
Fig. 6. An impression of an arrangement for counterclockwise spirals on the surface of the cone. The thick lines are the departure curves. The thin dashed lines are spiral segments that delimit trapezoidal regions that have the same next departure curve or obstacle region. The gray areas depict obstacle regions of other cones penetrating the surface. One of them cuts a departure curve into two collision-free intervals.

7.2. Datastructure

In order to efficiently perform the operations mentioned above, we create, prior to executing the algorithm, two *arrangements* (de Berg et al. 2000) for each cone on the surface of the cone; one for counterclockwise spirals and one for clockwise spirals. The arrangement for counterclockwise spirals contains all rl- and rr-departure curves, and the arrangement for clockwise spirals contains the lr- and ll-departure curves. (We only put actual departure curves in the arrangements; no arrival curves.) In both arrangements, we include the curves that delimit the cone's obstacle regions.

Next, we compute the *trapezoidal map* (de Berg et al. 2000) of each arrangement, where the sides of the trapezoids are *spiral segments*. This trapezoidal map partitions the surface of the cone into cells whose points have the same next departure curve or obstacle region. More precisely, if two spirals start at two points from the same cell of the trapezoidal map, the departure curve or the obstacle region the spirals next encounter is the same. Figure 6 shows an impression of an arrangement and its trapezoidal map for counterclockwise spirals.

Trapezoidal maps can be efficiently queried for point-location. That is, given a point on the surface of the cone, it returns the cell of the trapezoidal map in which this point is located. So, determining in which departure curve interval an edge arrives is easy. As we have seen above, each node (let it be an interval of departure curve $DC(C_i, C_j)$) has two outgoing edges. For the first edge, which stays on C_i, we have to determine the next departure curve that is encountered if we

Algorithm 2 TIMEMINIMALPATHEFFICIENTRESTRICTED(s, g)

```
1:  /* Initialize time values of departure curve intervals and priority queue */
2:  for all departure curve intervals $I$ do
3:      $I$.time ← $\infty$.
4:  for all straight-line segments from $s$ arriving at a tangency point $(q, t)$ on a cone do
5:      if the straight-line segment from $(s, 0)$ to $(q, t)$ is collision-free then
6:          Follow the spiral about the cone from $(q, t)$ until it runs into either an obstacle region or a departure curve interval.
7:          if the spiral hits a departure curve interval $I$ at time $t'$ then
8:              $I$.time ← $t'$.
9:              Insert $I$ into the priority queue $Q$
10:
11: /* Find time-minimal path */
12: while $Q$ is not empty do
13:     Pop element $I$ with minimal $I$.time from the queue.
14:     if the goal position is not collision-free anymore at time $I$.time then
15:         Path does not exist. Terminate.
16:     else if $I$ is the goal $g$ then
17:         Time-minimal path found! Terminate.
18:     else
19:         Let $DC(C_i, C_j)$ be the departure curve of which $I$ is an interval.
20:         Let $(q, t)$ be the point on $I$ with time value $I$.time
21:
22:         /* Handle outgoing edge 1 */
23:         Follow the spiral about cone $C_i$ from $(q, t)$ until it runs into either an obstacle region or a departure curve interval.
24:         if the spiral hits a departure curve interval $I'$ at time $t'$ then
25:             if $t' < I'$.time then
26:                 Remove $I'$ from $Q$ if it is in the priority queue.
27:                 $I'$.time ← $t'$.
28:                 Insert $I'$ into the priority queue $Q$
29:
30:         /* Handle outgoing edge 2 */
31:         Compute the bitangent straight-line segment leaving from $(q, t)$ on $I$ on the surface of cone $C_i$ arriving at the surface
            of cone $C_j$ at point $(q', t')$.
32:         if the straight-line segment from $(q, t)$ to $(q', t')$ is collision-free then
33:             Follow the spiral about cone $C_j$ from $(q', t')$ until it runs into either an obstacle region or a departure curve interval.
34:             if the spiral hits a departure curve interval $I'$ at time $t''$ then
35:                 if $t'' < I'$.time then
36:                     Remove $I'$ from $Q$ if it is in the priority queue.
37:                     $I'$.time ← $t''$.
38:                     Insert $I'$ into the priority queue $Q$
39:
40: Path does not exist.
```

proceed by moving along the spiral about C_i (line 23 in Algorithm 2). This can be done by querying the trapezoidal map on C_i for the point where we arrived on $DC(C_i, C_j)$. For the second edge, which traverses to C_j, we have to determine what the first departure curve is we will encounter there (line 33 in Algorithm 2). This can be done efficiently using the trapezoidal map we have computed on that cone. Using a point-location query, we can determine in what cell of the trapezoidal map the straight-line segment has arrived, and 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 (line 32 in Algorithm 2). Spiral segments may collide with other cones if these penetrate the spiral's cone surface. Since obstacle regions are incorporated into the arrangement, such collisions are inherently detected. Straight-line segments, however, may collide with any cone. Simply testing them for intersections against all cones would take $O(n)$ time per edge. We can do better, though, if for each departure curve and each cone we compute the *shadow interval* this cone casts on the departure curve. A shadow interval a cone C_k casts on a departure curve $DC(C_i, C_j)$ on cone C_i is defined as those points along $DC(C_i, C_j)$ from which a departure does not result in a smooth arrival on the surface of C_j, but in a collision with C_k. For each departure curve, all shadow intervals the cones cast are sorted into a list, such that it can be queried efficiently.

So, when we are considering the outgoing edge of a departure curve interval on $DC(C_i, C_j)$ that traverses to cone C_j, we first query the shadow interval list stored with $DC(C_i, C_j)$ for the point where we arrived on $DC(C_i, C_j)$ to test whether the outgoing edge collides with any cone.

7.3. Complexity Analysis

The complexity of Dijkstra's algorithm is well known to be $O(N \log N + E)$, where N is the number of nodes and E is the number of edges. The total number of nodes N (that is, departure curve intervals) can be bound as follows. 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 $O(n)$ collision-free intervals, as it is possibly intersected by $O(n)$ obstacle regions (each induced by one of the $O(n)$ cones), and each can split a departure curve into at most $O(1)$ segments. Hence, the total number of departure curve intervals, or nodes, is $O(n^3)$.

As each departure curve interval has $O(1)$ outgoing edges, the total number of edges is also $O(n^3)$. However, each edge requires some additional work when it is encountered during the algorithm. First, we have to find the departure curve interval in which it will arrive, and second, we must determine whether the edge is collision-free. Using the datastructure we constructed, these operations can be performed in $O(\log n)$ time. This gives us a total running time of $O(n^3 \log n)$ for our Dijkstra algorithm.

The total complexity of the constructed datastructures is as follows. The complexity of each arrangement is $O(n^2)$, as it contains $O(n)$ features (there are $O(n)$ departure curves on each cone, and $O(n)$ obstacles regions of other cones) of which each pair can only intersect $O(1)$ times. Computing each arrangement and its trapezoidal map can be done in $O(n^2 \log n)$ time using a sweep-line algorithm if all curves in the arrangement are x-monotone, or in our case, Θ-monotone (de Berg et al. 2000). This is actually the case, as both the departure curves and the curves delimiting the obstacle regions are functions from Θ to T when the growth rates of the disks are equal. Hence, each of the arrangements and their trapezoidal maps can be constructed in $O(n^2 \log n)$ time. As there are $O(n)$ cones, this step takes $O(n^3 \log n)$ time in total. The trapezoidal maps allow a point-location query to be performed in $O(\log n)$ time (de Berg et al. 2000).

For each departure curve, there are $O(n)$ cones that cast shadows. As each cone casts $O(1)$ shadow intervals, there are $O(n)$ intervals per departure curve. Sorting them takes $O(n \log n)$ time. As there are $O(n^2)$ departure curves, computing all the shadow interval lists takes $O(n^3 \log n)$ time in total. As each list contains $O(n)$ items, querying a point on a departure curve for any collision takes $O(\log n)$ time using binary search.

We conclude that both the preprocessing of the datastructure and the actual Dijkstra algorithm take $O(n^3 \log n)$ time.

Corollary 13. *The algorithm to compute a time-minimal path amidst n growing disks with equal growth rates runs in $O(n^3 \log n)$ time.*

8. The General Case: Disks Have Different Growth Rates

In this section we discuss the general case where the disks may have different growth rates. We cannot devise an algorithm with a time bound expressed in the number of disks in this case, but we present an efficient heuristic algorithm. Using experimental results we show that this algorithm is fast in practice.

8.1. A Heuristic Algorithm

Let us first show why we cannot use the same approach as in the previous section for the general case. Let us look at what happens to the slope of the departure curves in this case (see Figures 4(a) and (c)). In the case where the arrival cone has a slower growth rate, paths following the departure curves (provably) have a speed that is smaller than or equal to V (see Figure 4(a)). However, in the case where the arrival cone has a faster growth rate (Figure 4(c)), it is clear that this is not the case. The departure curve DC_{rr} is horizontal at some point, meaning that the speed of a path following this departure curve is ∞ there. Hence, Lemma 12 does not hold in the general case, and as a result 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 arises in that this algorithm considers many branches in the search tree which we know will not lead to a time-minimal path. For instance, it lets the spirals circle 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 (see Figure 7(a)).

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 (Θ, T) on the

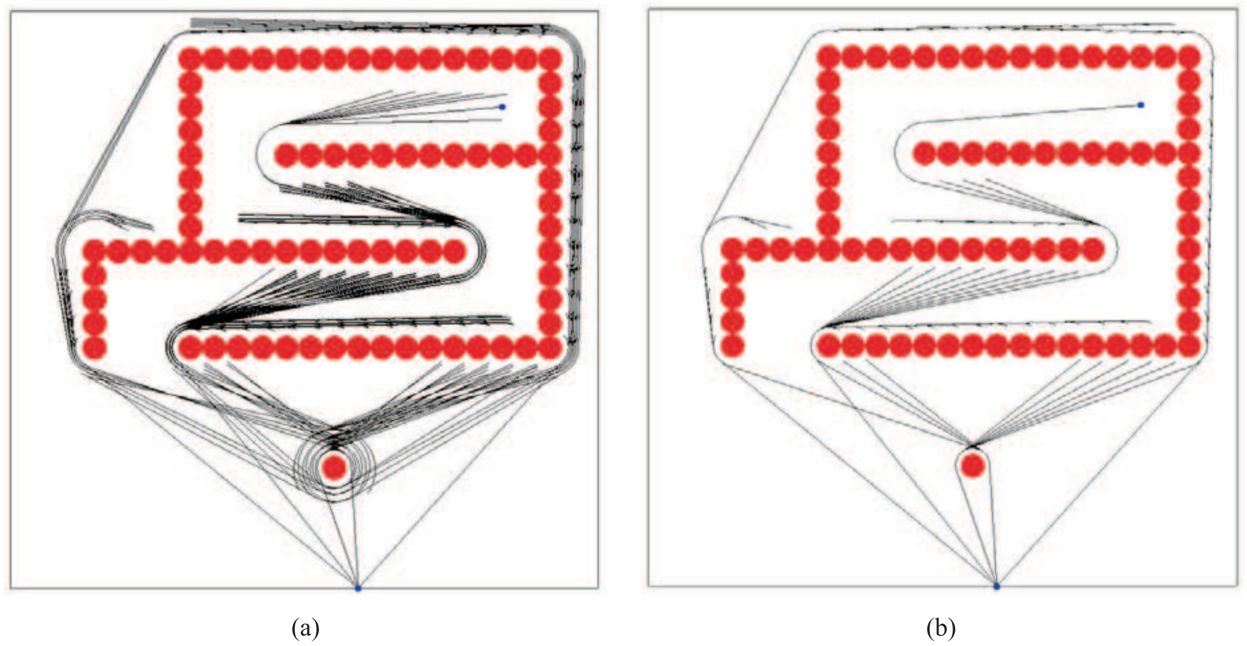

Fig. 7. An example problem in an environment with slowly growing disks. The start position is the dot on the bottom of the figure. The goal position is the dot in the upper right part of the figure. The figures show the disks with their initial radii. (a) The tree of explored paths using the basic algorithm presented in Section 6.1. (b) The tree of explored paths using the efficient heuristic algorithm presented in Section 8.

surface of cone C_i cannot be part of a time-minimal path if we have visited (Θ, T') already (where $T' < T$) *and* the vertical line segment on the surface of the cone between (Θ, T') and (Θ, T) is collision-free. This is because (Θ, T) is then in the wedge region of (Θ, T') (note that the speed along a vertical line segment is equal to the growth rate of the disk $v_i < V$). Hence a spiral encountering (Θ, T) need not be expanded any further.

To implement this practically, we only perform this test for a constant number of Θ. To this end, we augment Algorithm 1 by choosing a small constant ε, and drawing $2\pi/\varepsilon$ evenly distributed *vertical lines* on the surface of each cone. In Algorithm 3 we give the pseudocode of our heuristic algorithm. We maintain for each vertical line m one time-value at which it was last visited, say m.time. Given the order in which the points are considered in the priority queue, we know that when a point (q, t) is popped from the queue, it has a higher time-value than any point considered previously. So, if point (q, t) lies on vertical line m, its time-value t is larger than the time-value m.time of a point considered previously on that line. If the line segment between m.time and t on m is collision-free, (q, t) is in the wedge region of the point considered previously, and hence this point is not expanded (line 12 of Algorithm 3). However, if the vertical line segment between these two points is not collision-free, point (q, t) is the first to arrive in a new interval of the vertical line, and its outgoing edges must be inserted into the priority queue. From this moment on, t is set as the time value attached to the vertical line m, as we know that no point below t will be considered.

The smaller the value chosen for ε, the sooner the spirals can be pruned, and hence the smaller the size of the search tree. On the other hand, a smaller ε also causes the algorithm to perform more (costly) pruning 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 disks n or the value of ε, it turns out to be very fast in practice.

In Figure 7, we show an example environment with the search trees of both the basic algorithm presented in Section 6.1, and the efficient algorithm using the pruning heuristic. The disk close to the start position generates many branches in the search tree of the basic algorithm, as the spirals keep on winding around this obstacle. In the efficient algorithm these branches are pruned, and as a result, its search tree is much smaller than that of the basic algorithm.

8.2. A*

As shown in line 6 of Algorithm 3, we can replace the Dijkstra paradigm by an equally suited A* method (LaValle 2006) that

Algorithm 3 TIMEMINIMALPATHHEURISTICGENERAL(s, g)

```
for all vertical lines $m$ on all cones do
  $m$.visited $\leftarrow$ false.
for all endpoints $(q, t)$ of the collision-free segments from $s$ that are tangent to a cone do
  Insert $(q, t)$ into priority queue $\mathcal{Q}$.
while $\mathcal{Q}$ is not empty do
  Pop element $(q, t)$ from $\mathcal{Q}$ with minimal $t$ (Dijkstra), or with minimal $h(q, t)$ (A*).
  if the goal position is not collision-free anymore at time $t$ then
    Path does not exist. Terminate.
  else if $q = g$ then
    Time-minimal path found! Terminate.
  else
    if $(q, t)$ is not on a vertical line or $((q, t)$ is on a vertical line $m$ and ($m$ is not visited or the line segment between $t$ and
    $m$.time on $m$ is not collision-free)) then
      if $q$ is on a vertical line $m$ then
        $m$.visited $\leftarrow$ true.
        $m$.time $\leftarrow t$.
      $(q, t)$ is on the surface of a cone, say $C_i$, so proceed along the spiral about $C_i$ until it encounters an obstacle region,
      a departure curve, or a vertical line.
      if the spiral encounters a departure curve, say $DC(C_i, C_j)$, then
        $(q', t') \leftarrow$ the intersection point of the spiral and the departure curve.
        $(q'', t'') \leftarrow$ arrival point of the bitangent segment on the surface of $C_j$.
        Insert $(q', t')$ into $\mathcal{Q}$.
        if straight-line segment from $(q', t')$ to $(q'', t'')$ is collision-free then
          Insert $(q'', t'')$ into $\mathcal{Q}$.
      else if the spiral encounters a vertical line then
        $(q', t') \leftarrow$ the intersection point of the spiral and the vertical line.
        Insert $(q', t')$ into the priority queue $\mathcal{Q}$.
Path does not exist.
```

is faster in practice as it focuses the search to the goal. To this end, we use a heuristic value $h(q, t)$ as the key-value by which the elements (q, t) are sorted in the priority queue.

In order for this heuristic to be *admissible*, the heuristic value $h(q, t)$ must be a *lower bound* estimate of the arrival time at the goal of the path through (q, t). Also, in order for the algorithm to remain correct, we have to make sure that the A* heuristic does not invalidate the premise that among the points on the same vertical line, the point with the smallest time-value is always considered first.

The trivial heuristic $h(q, t) = t$, which makes the A* algorithm equivalent to Dijkstra's algorithm, obeys both of the above requirements. However, we can show that the more informed heuristic that uses the Euclidean distance to the goal divided by the maximum speed, i.e.

$$h(q, t) = t + \frac{\|g - q\|}{V}, \tag{11}$$

is valid as well. Obviously, this heuristic is an admissible lower bound estimate. The following lemma shows that it also obeys the second requirement.

Lemma 14. *Let (q, t) and (q', t') with $t < t'$ be two points on the same vertical line on the surface of a cone C_i with growth rate v_i. Then $h(q, t) < h(q', t')$ where $h(q, t)$ is defined as in Equation (11).*

Proof. As (q, t) and (q', t') are on the same vertical line, we know that $\|q - q'\| = v_i(t' - t) < V(t' - t)$. Hence, by the triangle inequality:

$$\begin{aligned} h(q, t) &= t + \frac{\|g - q\|}{V} \leq t + \frac{\|g - q'\| + \|q - q'\|}{V} \\ &< t + \frac{\|g - q'\| + V(t' - t)}{V} \\ &= t' + \frac{\|g - q'\|}{V} = h(q', t'). \end{aligned}$$

□

Hence, the heuristic of Equation (11) can be used safely in our algorithm.

Table 1. Results (in Seconds) for the Environment of Figure 7 (82 Obstacles).

	ε					
Algorithm	$2\pi/10$	$2\pi/20$	$2\pi/40$	$2\pi/60$	$2\pi/100$	$2\pi/1{,}000$
Basic (Dijkstra)	5.57	5.58	5.61	5.73	5.74	6.04
Heuristic (Dijkstra)	0.18	0.15	0.16	0.16	0.16	0.17
Heuristic (A*)	0.12	0.13	0.14	0.14	0.14	0.15

Table 2. Results (in Seconds) for the Environment of Figures 9 and 10 (10 Obstacles).

	ε					
Algorithm	$2\pi/10$	$2\pi/20$	$2\pi/40$	$2\pi/60$	$2\pi/100$	$2\pi/1{,}000$
Basic (Dijkstra)	0.0087	0.0091	0.0092	0.0096	0.0097	0.0219
Heuristic (Dijkstra)	0.0056	0.0060	0.0066	0.0069	0.0078	0.0189
Heuristic (A*)	0.0028	0.0026	0.0028	0.0029	0.0031	0.0084

8.3. Implementation Details

We implemented Algorithm 3 in C++. Here, we discuss some of the details of the implementation.

8.3.1. Finding Intersections of Spirals and Departure Curves

Among the outgoing edges of a point (q, t) is a spiral segment to the next vertical line or, in cases where this spiral segment crosses a departure curve, a spiral segment to the intersection point with the departure curve (see lines 15–24 of Algorithm 3). Hence, we have to find the intersection points of the departure curves and a segment of a spiral between q and the next vertical line. In our implementation, these intersections are found using a combination of two approximate root-finding algorithms (Burden and Faires 2001). We must note that if between two consecutive vertical lines the spiral intersects the *same* departure curve twice (which is theoretically possible), only one intersection will be found by our root-finding algorithm. In order to have a correctly functioning algorithm that does not "miss" departure curves, we rely on a small value of ε (note that this is not a limitation of the algorithm, but rather a limitation of our implementation), so that the spiral segments between two vertical lines are short. The probability that an intersection is missed is negligible for sufficiently small values of ε.

8.3.2. Collision Checking

Checking straight-line segments for collisions is done by testing them for intersections with all cones, except those they are tangent to. Checking whether the spiral segments between two consecutive vertical lines run into an obstacle region is done by approximating them by one or more small straight-line segments, and testing them for intersections with all cones, except the one this spiral is on. In our implementation, we use a single straight-line segment to approximate the spiral segments, as we assume that the radial distance ε between two consecutive vertical lines is small.

8.4. Experimental Results

As a proof of concept, we created an interactive application for planning paths amidst growing disks. The properties of the growing disks (position, size, growth rate) can be changed by the user, and a new path is computed on-the-fly. From this application we report results that give an indication of the running time of the heuristic algorithm versus the basic algorithm, show the effect of parameter ε, and indicate how the performance scales with a growing number of disks. Experiments were run on a Pentium IV 3.0 GHz with 1 GB of memory.

In our first experiment, we show the difference in running time of the basic algorithm (Algorithm 1) and the heuristic algorithm (Algorithm 3) with both the Dijkstra and the A* paradigm. We do these comparisons for different values of ε (note that we also use the ε parameter for the basic algorithm, as our implementation relies on ε for collision checking and root finding (see Section 8.3)). We ran the experiment on both the environment of Figure 7 containing 82 obstacles and the environment of Figures 9 and 10 containing 10 obstacles. The results are given in Tables 1 and 2, respectively.

From the results, we can first of all see that the value of ε has little effect on the running time of the algorithm. In some cases we see that for $\varepsilon = 2\pi/10$ the running time is higher than for $\varepsilon = 2\pi/20$, as a larger value for ε causes less useless

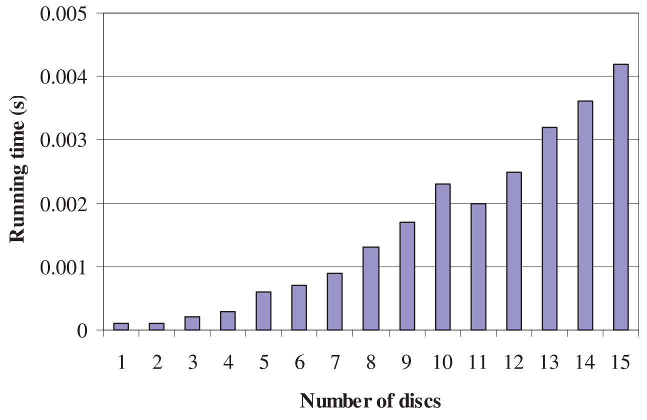

Fig. 8. The running time of the heuristic algorithm (using the A* paradigm) for a growing number of disks.

branches of the tree to be pruned. Other than that, we see that the running time (very) slowly increases with a smaller value of ε.

Second, we see that the heuristic algorithm is much faster than the basic algorithm in the environment of Figure 7. On average, the difference is a factor of 35 in this case. This number is so high mainly because the environment was designed specifically as an almost worst-case example for the basic algorithm, as can be seen in Figure 7. The effect of applying the A* paradigm rather than the Dijkstra paradigm in the heuristic algorithm is rather modest in this case, but it does provide an additional speed-up.

In the more "typical" environment of Figures 9 and 10, the difference in running time between the basic algorithm and the heuristic algorithm is much smaller. This is because the environment does not allow the spirals to wind around the obstacles multiple times (thereby creating many branches in the search tree) before it hits an obstacle region on the surface of a cone. Still, the heuristic algorithm is roughly 1.5 times faster than the basic algorithm. Applying the A* paradigm gives a significant additional speed-up of about a factor of 2 in this case. This number is relatively high mainly because the final time-minimal path leads more or less directly towards the goal (see Figure 9), in which case the A* heuristic to focus the search toward the goal works most effectively[7] (this is much less the case in the environment of Figure 7).

In the second experiment, we measure how the performance of the algorithm scales with the number of obstacles. We performed experiments in "typical" environments of the type shown in Figures 9 and 10, in which the disks are more or less randomly scattered over the environment. We vary the number of disks between 1 and 15. We did not perform experiments for more than 15 disks, as it appeared to be difficult to find sensible setups with this many disks that still contain a valid path to the goal. In each experiment, we manually chose the configuration of the disks such that only a difficult, yet valid path to the goal exists. (By difficult, we mean a path visiting many growing disks before the goal is reached.) The value of ε was chosen sufficiently small and fixed at $2\pi/40$ (we note that a too large value of ε may cause our implementation to produce invalid results, as our implementation relies on a small ε for accurate collision checking and root finding; see Section 8.3).

For each experiment, we averaged the running times over ± 5 different queries, by varying the start and the goal position. We manually chose the set of queries to average over, such that: (1) a path to the goal exists, (2) the resulting paths are topologically different from each other, and (3) the running time was among the highest of all possible queries. For instance, we did not choose the start and goal configuration very close to each other, such that the time-minimal path would be a direct connection between them (the time it takes to compute such a path is negligible). In this way, we try to give a fair idea of the performance of our algorithm. The results are given in Figure 8.

From the results, we can see that our implementation is fast. Even for 15 growing disks, the running time is only 0.0042 seconds. From the figure it seems that the running time is more or less quadratically related to the number of disks. This is what we expected based on the implementation. The running time of the algorithm does not only depend on the number of obstacles, but also on the exact configuration of the disks, and how well the A* method manages to focus the search, etc. This may explain why the algorithm performed slightly better for 11 disks than for 10 disks.

In Figure 9, snapshots are shown of a time-minimal path amidst 10 growing disks. Figure 10 gives a three-dimensional view of the same environment.

9. Conclusion

In this paper we presented an algorithm for computing time-minimal paths (minimum time paths) amidst disks that grow over time. For the restricted case where all disks have the same growth rate, we have given an $O(n^3 \log n)$ algorithm, where n is the number of obstacles. We have left this problem open for the general case where disks can have different growth rates, but we presented a fast implementation of a heuristic algorithm that works well in practice.

The research presented in this paper was motivated by motion planning in unpredictable dynamic environments, as a growing disk can model the region that is guaranteed to contain a moving obstacle of which the maximum speed 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

7. This is because the lower bound estimates of the total path duration used in the A* algorithm are more accurate when the final path leads more directly towards the goal.

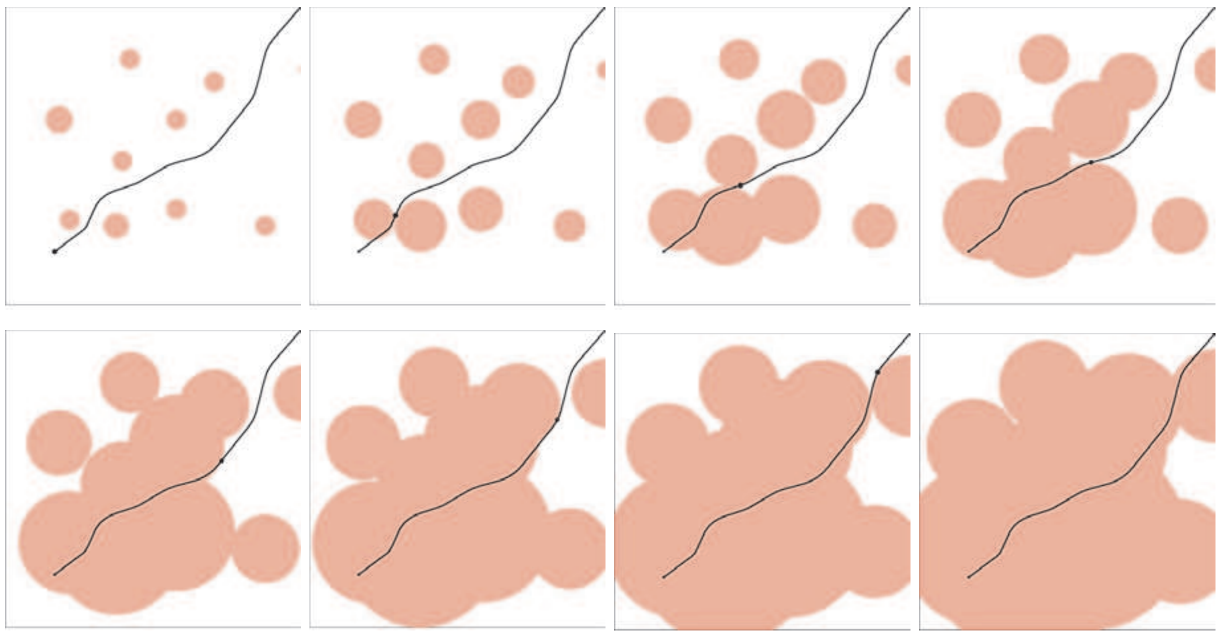

Fig. 9. A time-minimal path amidst 10 growing disks. A small dot indicates the position along the path at $t = 0, 1, \ldots, 7$. The pictures were generated by our application.

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. Results have shown that such paths can be generated very quickly. A clear limitation of our approach remains that the robot is required to have a higher maximum speed than any of the moving obstacles, but a great advantage over other methods is that the planned paths are guaranteed to be safe.

An interesting direction for future work would be to actually implement our algorithm in a real-time online planner. We briefly give some leads here as to how our algorithm can be applied in a continuous cycle of sensing and planning of the type discussed in the introduction.

When we start planning at the beginning of a planning cycle of duration τ, say at time t, we use the following input for our algorithm: we take the position the robot will have at time $t + \tau$ from the path the robot is currently following, and use it as the start position. Further, we take all obstacle positions we observe at time t, and use it as the center positions of the growing disks. Finally, as the initial radius of the growing disks, we take the radius of the moving obstacle, and add to this the product of τ and the obstacle's maximum speed, to compensate for its motion between time t and time $t + \tau$. When the planning has finished at the end of the planning cycle, the computed path is ready for use in the next planning cycle starting at time $t + \tau$.

During the entire process of continuous replanning, the robot is guaranteed to be safe, as all paths that are computed are guaranteed to be collision-free. Also, we are allowed to choose a conveniently large value for τ. It should at least be as large as the time it takes for our algorithm to plan a path. Our implementation shows that such paths can be generated in less than 0.01 seconds, which is well within real-time requirements.

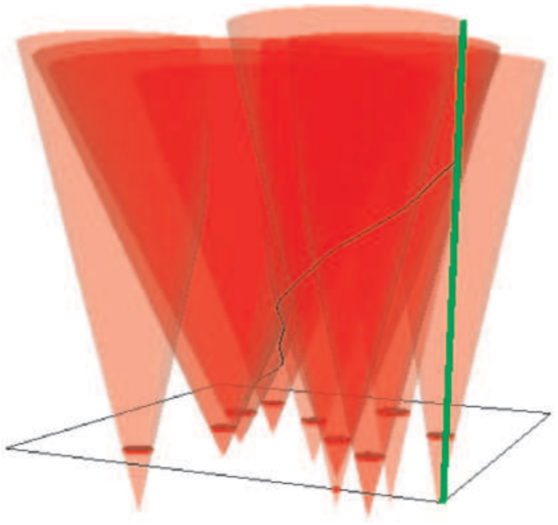

Fig. 10. The three-dimensional view of the same path as in Figure 9.

A possible complication that might arise is that a guaranteed safe path to the goal often does not exist. This occurs when the goal is covered by a growing disk before it can be reached. However, in a real-time scheme of continuous replanning, this may not be so much of a problem, as the paths need only be safe for at least τ time. Such a path does exist in most cases, and the planner might choose the path that comes closest to the goal among those it was able to explore. It seems that this can easily be incorporated into our algorithm. (If a path that is safe for τ time does not exist, any planner would be in serious trouble, and a collision seems unavoidable.)

Another possible issue is that our algorithm is not 100% guaranteed to finish planning within τ time, even though our planner is fast, and even if τ is chosen large. Although it is unlikely that this will occur, it is easily remedied. When planning time has run out, the algorithm simply selects the path from the search tree that leads to the currently most promising entry in the priority queue. As a consequence, this path does not go all the way to the goal, but as we have seen above, this is not a problem as long as the path is safe for at least τ time.

Acknowledgments

The authors like to thank Jack Snoeyink for valuable comments that improved the quality of this paper. 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).

Appendix A: Expressions of Equation (9)

The expressions for $a(\Theta, T)$ and $D(\Theta, T)$ in Equation (9) are as follows:

$$\begin{aligned}
a(\Theta, T) &= p_{iy}\sqrt{V^2-1}\cos\Theta + p_{iy}\sin\Theta \\
&+ p_{ix}\cos\Theta - p_{ix}\sqrt{V^2-1}\sin\Theta + r_i v_i + (V^2-1)T \\
D(\Theta, T) &= 2r_i v_i V^2 T + 2p_{iy}^2\sqrt{V^2-1}\cos\Theta\sin\Theta \\
&+ 2p_{ix}\sqrt{V^2-1}T\sin\Theta - 2p_{iy}\sqrt{V^2-1}T\cos\Theta \\
&+ 2p_{iy}r_i v_i\sin\Theta + 4p_{iy}p_{ix}\sin\Theta\cos\Theta \\
&+ 2p_{iy}V^2T\sin\Theta + 2p_{ix}r_i v_i\cos\Theta \\
&- 2p_{ix}^2\sqrt{V^2-1}\cos\Theta\sin\Theta + 2p_{ix}V^2T\cos\Theta \\
&- 2p_{iy}p_{ix}\sqrt{V^2-1} + v_i^2V^2T^2 + p_{iy}^2V^2\cos^2\Theta \\
&- 2r_i v_i T - 2p_{ix}T\cos\Theta - 2p_{iy}T\sin\Theta \\
&+ 4p_{ix}p_{iy}\sqrt{V^2-1}\cos^2\Theta + 2p_{ix}^2\cos^2\Theta - V^2T^2 \\
&+ p_{iy}^2v_i^2 + p_{ix}^2v_i^2 - v^2T^2 + r_i^2V^2 - p_{iy}^2V^2 \\
&- 2p_{iy}^2\cos^2\Theta + T^2 + 2p_{iy}v_i^2\sqrt{V^2-1}T\cos\Theta \\
&+ 2p_{iy}r_i v_i\sqrt{V^2-1}\cos\Theta - 2p_{ix}r_i v_i\sqrt{V^2-1}\sin\Theta \\
&- 2p_{ix}v_i^2\sqrt{V^2-1}T\sin\Theta - 2p_{ix}p_{iy}V^2\sin\Theta\cos\Theta \\
&+ p_{iy}^2 - p_{ix}^2 - p_{ix}^2V^2\cos^2\Theta
\end{aligned}$$

Appendix B: Proof of Lemma 12

We have to prove that in cases where the growth rate is equal for all disks, a path following any departure curve on the surface of any cone has a speed less than or equal to V. Let us consider the departure curves on a cone C, of which we assume, without loss of generality, that the disk associated with C has radius 0 at $t = 0$, that the disk is centered at the origin, and that the disk grows with speed 1 (other configurations can be transformed such that these conditions hold). Note that $V > 1$.

We first consider the departure curves leading to another cone, and later discuss the departure curves leading to the goal position.

B.1. Departure Curves Leading to Another Cone

Let us consider the departure curves leading to cone C_i, with initial radius r and speed 1 (the same as C). We assume, without loss of generality, that the center of C_i is located on the positive y-axis at $(0, p)$ (other configurations can be rotated such that this holds). We only consider the departure curves $DC_{rr}(C, C_i)$ and $DC_{rl}(C, C_i)$ in this proof (the proof for $DC_{ll}(C, C_i)$ and $DC_{lr}(C, C_i)$ is equivalent, as they are symmetric to $DC_{rr}(C, C_i)$ and $DC_{rl}(C, C_i)$).

Proving that the departure curve $DC_{rr}(C, C_i)$ has a speed less than V is trivial. This departure curve is a vertical-line in the ΘT-coordinate frame (see Figure 4(b)), with equation

$$\Theta = \arctan\left(-\frac{r}{p}, \frac{\sqrt{p^2 - r^2}}{p}\right).$$

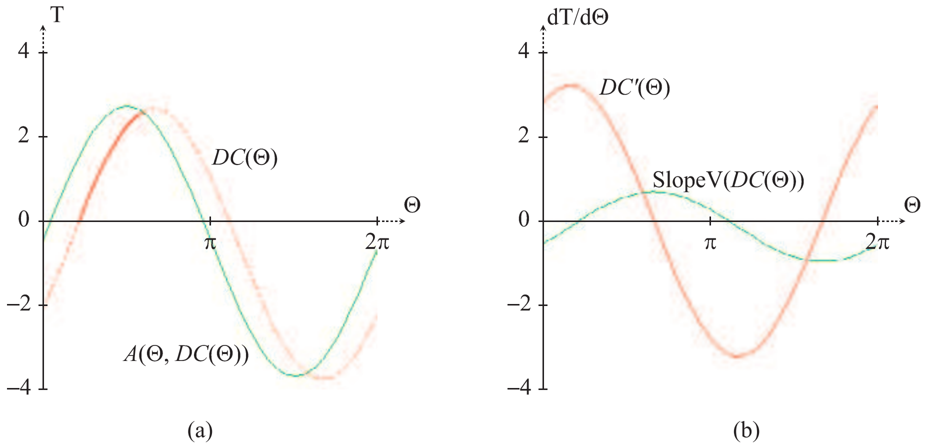

Fig. 11. (a) The functions $DC(\Theta)$ and $A(\Theta, DC(\Theta))$ for $p = 6$, $r = 1$, and $V = 4$. The part of $DC(\Theta)$ that is an actual departure curve is drawn thick. The parts that can be ignored are dashed. Here $A(\Theta, DC(\Theta))$ gives the arrival time at cone C_i when left from the departure curve at angle Θ. (b) The function $\text{SlopeV}(DC(\Theta))$, and the derivative $DC'(\Theta)$ of $DC(\Theta)$ for $p = 6$, $r = 1$, and $V = 4$. Here $\text{SlopeV}(DC(\Theta))$ gives the slope corresponding to V for the T-value of $DC(\Theta)$ at angle Θ.

Hence, the speed when moving along this 'curve' is 1, i.e. the same as the growth rate of the disk. As $V > 1$, the speed along this departure curve is less than V.

The case for $DC_{rl}(C, C_i)$ is less easy. In Section 5.1 we have seen how we can derive equations for the departure curves. If we work out the mathematics, that is, we solve $DC(\Theta, T) = 0$ for T, $DC_{rl}(C, C_i)$ appears to have the following equation:

$$\begin{aligned} DC(\Theta) &= -\frac{p}{\sqrt{V^2-1}}\cos\Theta + \frac{p(V^2-2)}{2(V^2-1)}\sin\Theta \\ &\quad - \frac{rV^2}{2(V^2-1)}. \end{aligned} \tag{12}$$

The arrival time $A(\Theta, DC(\Theta))$ at cone C_i when leaving from the departure curve at $(\Theta, DC(\Theta))$ is

$$A(\Theta, DC(\Theta)) = \frac{pV^2}{2(V^2-1)}\sin\Theta - \frac{r(V^2-2)}{2(V^2-1)}.$$

We plotted these functions for $p = 6$, $r = 1$ and $V = 4$ (see Figure 11(a)). We only have to prove the speed bound for the part of the curve that is actually a departure curve (thick portion in the figure). This is where the arrival time is larger than the departure time (we can also ignore the part of the curve where $T < 0$, but that is not necessary for this proof).

Let us now look at the slope along the departure curve, that is, we take the derivative $dT/d\Theta$ of $DC(\Theta)$:

$$DC'(\Theta) = \frac{p}{\sqrt{V^2-1}}\sin\Theta + \frac{p(V^2-2)}{2(V^2-1)}\cos\Theta. \tag{13}$$

Note that the smaller the slope, the higher the actual speed is. The slope which corresponds to speed V varies with T. We will now derive the function for the slope corresponding to V, given a T-value. A spiral on the cone has speed V. Its equation is

$$\Theta_{\text{spiral}}(T) = \sqrt{V^2-1}\log T,$$

as we know from Equation (5). The derivative of this function is

$$\Theta'_{\text{spiral}}(T) = \frac{\sqrt{V^2-1}}{T}.$$

To switch from $d\Theta/dT$ to $dT/d\Theta$, and have the actual slope in the ΘT-coordinate frame corresponding with the maximum speed V, we take the reciprocal of the above, giving

$$\text{SlopeV}(T) = \frac{T}{\sqrt{V^2-1}}. \tag{14}$$

To prove that the departure curve has a speed less than V for the part of the curve for which the arrival time is larger than the departure time, we have to prove that (recall that the larger the slope, the smaller the speed is)

$$\begin{aligned} A(\Theta, DC(\Theta)) &\geq DC(\Theta) \\ \Rightarrow DC'(\Theta) &\geq \text{SlopeV}(DC(\Theta)). \end{aligned}$$

In Figure 11(b), we have plotted both $\text{SlopeV}(DC(\Theta))$ and $DC'(\Theta)$ for $p = 6$, $r = 1$, and $V = 4$.

As we can already conjecture from Figure 11, it seems that not only does the above implication hold, but that it is actually an equivalence. Indeed, if we solve $A(\Theta, DC(\Theta)) = DC(\Theta)$ and $\text{SlopeV}(DC(\Theta)) = DC'(\Theta)$ for Θ, we find the exact same solutions. However, to actually prove that the above is an equivalence, we perform sign analysis on the functions $DC'(\Theta) - \text{SlopeV}(DC(\Theta))$ and $A(\Theta, DC(\Theta)) - DC(\Theta)$, assuming that $V > 1$, $r \geq 0$, and $p > r$. The sign of both of these functions evaluate to the same[8]:

$$\begin{aligned} & \text{signum}(DC'(\Theta) - \text{SlopeV}(DC(\Theta))) \\ = \; & \text{signum}(A(\Theta, DC(\Theta)) - DC(\Theta)) \\ = \; & \text{signum}(p\sqrt{V^2-1}\sin\Theta + p(V^2-1)\cos\Theta \\ + \; & r\sqrt{V^2-1}). \end{aligned}$$

Hence, for the part of the departure curve where the arrival time is larger than the departure time, the speed along the departure curve is less than or equal to V.

B.2. Departure Curves Leading to the Goal Position

To prove the same for departure curves leading to the goal position, we follow the exact same strategy. We assume, without loss of generality, that the goal position is located at the positive y-axis at $(0, g)$. In this case, the equations for the departure curve and the arrival time become

$$\begin{aligned} DC(\Theta) &= -\frac{g(\cos\Theta - \sqrt{V^2-1}\sin\Theta)}{\sqrt{V^2-1}}, \\ A(\Theta, DC(\Theta)) &= g\sin\Theta. \end{aligned}$$

The slope along $DC(\Theta)$ is

$$DC'(\Theta) = \frac{g(\sin\Theta + \sqrt{V^2-1}\cos\Theta)}{\sqrt{V^2-1}},$$

and the equation for SlopeV is the same as in Equation (14). The sign of both functions $DC'(\Theta) - \text{SlopeV}(DC(\Theta))$ and $A(\Theta, DC(\Theta)) - DC(\Theta)$ evaluate to the same:

$$\begin{aligned} & \text{signum}(DC'(\Theta) - \text{SlopeV}(DC(\Theta))) \\ = \; & \text{signum}(A(\Theta, DC(\Theta)) - DC(\Theta)) \\ = \; & \text{signum}(\cos\Theta). \end{aligned}$$

Hence, also for the departure curves leading to the goal position, the speed along the departure curve is less than or equal to V when the arrival time is larger than the departure time.

8. signum(x) is defined to be -1 if $x < 0$, 0 if $x = 0$, and 1 if $x > 0$.

References

Bekris, K. E. and Kavraki, L. E. (2007). Greedy but safe replanning under kinodynamic constraints. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 704–710.

Burden, R. L. and Faires, J. D. (2001). *Numerical Analysis*, 7th edn. Pacific Grove, CA, Brooks/Cole.

Chang, E. C., Choi, S. W., Kwon, D. Y., Park, H. and Yap, C. K. (2005). Shortest path amidst disc obstacles is computable. *Proceedings of the Annual Symposium on Computational Geometry*, pp. 116–125.

de Berg, M., van Kreveld, M., Overmars, M. H. and Schwarzkopf, O. (2000). *Computational Geometry: Algorithms and Applications*, 2nd edn. Berlin, Springer.

Fiorini, P. and Shiller, Z. (1998). Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, **17**(7): 760–772.

Fraichard, T. and Asama, H. (2004). Inevitable collision states—a step towards safer robots? *Advanced Robotics*, **18**(10): 1001–1024.

Frazzoli, E., Dahleh, M. A. and Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *Journal of Guidance, Control and Dynamics*, **25**(1): 116–129.

Fujimura, K. and Samet, H. (1993). Planning a time-minimal motion among moving obstacles. *Algorithmica*, **10**(1): 41–63.

Hsu, D., Kindel, R., Latombe, J. and Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, **21**(3): 233–255.

Jaillet, L. and Simeon, T. (2004). A PRM-based motion planner for dynamically changing environments. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1606–1611.

Kallman, M. and Mataric, M. (2004) Motion planning using dynamic roadmaps. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 4399–4404.

Latombe, J. C. (1991). *Robot Motion Planning*. Boston, MA, Kluwer.

LaValle, S. M. (2006) *Planning Algorithms*. New York, Cambridge University Press.

Leven, P. and Hutchinson, S. (2002). A framework for real-time path planning in changing environments. *The International Journal of Robotics Research*, **21**(12): 999–1030.

Mitchell, J. S. B. (2000). Geometric shortest paths and network optimization. *Handbook of Computational Geometry*. Amsterdam, Elsevier, pp. 633–701.

Petty, S. and Fraichard, T. (2005). Safe motion planning in dynamic environments. *Proceedings of the IEEE/RSJ In-*

ternational Conference on Intelligent Robots and Systems, pp. 3726–3731.

Stentz, A. (1995). The focussed D* algorithm for real-time replanning. *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 1652–1659.

van den Berg, J., Ferguson, D. and Kuffner, J. (2006). Anytime path planning and replanning in dynamic environments. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2366–2371.

Vannoy, J. and Xiao, J. (2006). Real-time planning of mobile manipulation in dynamic environments of unknown changes. *Proceedings of Robotics: Science and Systems*.

Vasquez, D., Large, F., Fraichard, T. and Laugier, C. Highspeed autonomous navigation with motion prediction for unknown moving obstacles. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 82–87.

Zucker, M., Kuffner, J. and Branicky, M.(2007) Multipartite RRTs for rapid replanning in dynamic environments. *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1603–1609.