Real-Time A* Search With Depth-k Lookahead*

Emre Sefer[1] **Ugur Kuter**[2] **Dana Nau**[1,3,2]
[1]Department of Computer Science, [2]Institute for Advanced Computer Studies,
and [3]Institute for Systems Research
University of Maryland, College Park, Maryland 20742, USA
{esefer,ukuter,nau}@cs.umd.edu

Abstract

We consider real-time planning problems in which some states are *unsolvable*, i.e., have no path to a goal. Such problems are difficult for real-time planning algorithms such as RTA* in which all states must be solvable.

We identify a property called k-*safeness*, in which the consequences of a bad choice become apparent within k moves after the choice is made. When k is not too large, this makes it possible to identify unsolvable states in real time.

We provide a modified version of RTA* that is provably complete on all k-safe problems, and we show that real-time deterministic versions of the well-known Tireworld and Racetrack domains are k-safe.

Introduction

There are many AI planning applications in which agents need to generate and execute plans in real time. Examples include UAVs (Weiss, Naderhirn, and del Re 2006), humanoid robots (Gutmann, Fukuchi, and Fujita 2005), real-time strategy games (Stene 2006), and RoboCup robots (Sherback, Purwin, and D'Andrea 2006).

In such applications there are situations where a bad choice of action may lead to failure. If a UAV ventures into a dangerous location, it may be shot down; or if a humanoid robot loses balance and falls down a stairway, it may be damaged; or if a player makes a bad move in a real-time strategy game or a RoboCup game, the opponents may achieve an unassailable advantage.

Consequently, it is important for a real-time planner to identify, before executing an action, whether the action may produce an *unsolvable state*, i.e., a state from which the goal is no longer reachable.

There are several methods and algorithms (Koenig 2001a) for solving real-time planning problems by interleaving planning and plan execution (Koenig 1999; Bulitko et al. 2007b), but they often require that the planning problem be *everywhere solvable*, i.e., the goal must be achievable at every state in the state space. Problems like the ones described above, in which some of the states are unsolvable, cause difficulty for such algorithms.

In offline planning, unsolvable states are avoided by searching all the way to the goal, thereby verifying that the goal is achievable at each state along the way. But in real-time planning, where unsolvable states need to be identified quickly, such an approach is infeasible because it can require exorbitant amounts of time (Erol, Nau, and Subrahmanian 1995).

In real-time planning, we believe there often are situations in which the unfortunate consequences of a bad choice become evident shortly after performing the action. Our objective is to provide a domain-independent way to identify and avoid such situations. Our contributions are as follows:

1. We define a property called k-*safeness*. If a problem is k-safe, this means that for every unsolvable state s that is reachable from the initial state, the longest simple path from s has length $\leq k$. Consequently, s's unsolvability can be detected by a depth-k lookahead.
2. We describe d-*lookahead* RTA*, which is similar to the well-known RTA* algorithm (Korf 1990) but has been modified to look ahead d steps before committing to its next action. We prove that d-lookahead-RTA* is complete on k-safe problems whenever $d \geq k$.
3. We analyze deterministic real-time versions of the well-known Tireworld (Littman and Younes 2004; Younes and Littman 2004) and Racetrack (Gardner 1986; Wikipedia 2009) domains. We show that given a map of a Tireworld or Racetrack domain, it is easy to derive a value k such that every planning problem on this map is k-safe.

Preliminaries

Let $G = (S, E, c, s_0, S_g)$ be a finite state space in which S is the set of states, E is the set of edges, c is the nonnegative cost function (see below), s_0 is the initial state, and S_g is the set of goal states. If there is a path from s_i to s_j then s_i is an *ancestor* of s_j and s_j is a *descendant* of s_i. If the path has length 1 then s_i is a *parent* of s_j and s_j is a *child* of s_i.

A state s is *solvable* if there is a path from s to a goal state; otherwise s is *unsolvable*. G is solvable if s_0 is solvable.

Above, c assigns a cost $c(s, s')$ to each edge $(c, c') \in E$. By extension, the cost of a path $\pi = \langle s_0, s_1, \ldots, s_n \rangle$ is $c(\pi) = \sum_{i=1}^{n} c(s_{i-1}, s_i)$. As usual, we let $h^*(s) =$

An earlier version of this paper has been accepted for presentation at the IJCAI-09 Workshop on Self- and Autonomous Systems (SAS): Reasoning and Integration Challenges (`http://www.laas.fr/SAS09`).

$\min\{c(\pi) \mid \pi \text{ solves } s\}$; hence if s is unsolvable then $h^*(s) = \infty$.

We take the *length* of a path to be the number of edges in the path. A *simple path* is one in which all states are different. A state s's *simple height*, $H(s)$, is the length of the longest (i.e., largest number of edges) simple path that begins at s. It follows immediately that $H(s) \geq 0$ for every state s.

The following lemma follows immediately:

Lemma 1. *A state s is solvable iff there is a path of length $\leq H(s)$ from s to a goal.*

Definition 2. Let G be solvable and $k \geq 0$ be an integer. Then G is *k-safe* iff for every state s that is reachable from s_0, if s is unsolvable then $H(s) \leq k$.

For example, consider the case where G is 0-safe. This means that every unsolvable state has simple height 0, i.e., for every unsolvable state s, either s is childless or else s has a single child, namely s itself.

d-Aware Search

If h is a heuristic function, we will say that h is *d-aware* iff $h(s) = \infty$ for every unsolvable state s of simple height $H(s) \leq d$. Intuitively, h is d-aware if it can detect unsolvable states that are $\leq d$ steps ahead.

If h is any admissible heuristic function for G and $d \geq 0$, it is easy to create from h a d-aware admissible heuristic function h^d, which we will call the *d-aware version* of h. The definition is as follows:

$$h^d(s) = \begin{cases} \infty, & \text{if } H(s) \leq d \text{ and there is no path of length} \leq d \text{ from } s \text{ to a goal state,} \\ h(s), & \text{if the above condition fails and } d = 0, \\ \min\{c(s,t) + h^{d-1}(t) : t \text{ is a child of } s\}, & \text{otherwise.} \end{cases} \quad (1)$$

Lemma 3. *If h is admissible, then h^d is both admissible and d-aware.*

Proof. Admissibility: Let s be any state of G. If $H(s) \leq d$ and there is no path of length $\leq d$ from s to a goal state (i.e., the first case of Eq. (1)), then it follows from Lemma 1 that s is unsolvable, whence $h^*(s) = \infty$; so $h^d(s) = \infty$ is admissible in this case. Otherwise if $d = 0$ (the second case of Eq. (1)), then $h^d(s) = h(s)$; and since $h(s)$ is admissible, so is $h^d(s)$. Otherwise the third case of Eq. (1) applies, and $h^d(s) \leq \min\{c(s,t) + h^{d-1}(t) : t \text{ is a child of } s\}$. It follows that if $h^{d-1}(t) \leq h^*(t)$ for every child t of s, then

$$h^d(s) \leq \min\{c(s,t) + h^*(t) : t \text{ is a child of } s\} = h^*(s).$$

Consequently it is easy to show by induction that $h^d(s) \leq h^*(s)$, so h^d is admissible in this case too.

d-awareness: Let s be any unsolvable state for which $H(s) \leq d$. Then it follows from Lemma 1 that there is no path of length $\leq d$ from s to a goal, whence $h^d(s) = \infty$. □

function $h^d(s)$: **return** $\bar{h}^d(s, \emptyset)$

function $\bar{h}^d(s, V)$:
1. if s is not a goal state and all of its children are in V then **return** ∞
2. if $d = 0$ then **return** $h(s)$
3. **return** $\min\{c(s,t) + \bar{h}^{d-1}(t, V \cup \{s\}) : t \text{ is a child of } s\}$

Figure 1: A simple implementation of h^d.

It is easy to implement h^d as shown in Fig. 1. If G is a tree with branching factor b, then h^d runs in time $O(b^d)$. If G is a graph, then the computation can sometimes be done more quickly by caching each state's value as it is computed, and using the cached value on subsequent visits to that state. For example, if there is a constant c such that there are at most c states at each search depth, then caching allows h^d to be computed in time $O(bd)$ rather than $O(b^d)$. Regardless of whether G is a tree or a graph, if there are fixed upper bounds on b and d, then h^d runs in time $O(1)$.

Definition 4. *d-lookahead-RTA** is the following modification of the RTA* algorithm: all calls to h are replaced with calls to h^d instead.

Equivalently, d-lookahead-RTA* runs RTA* with h^d as the heuristic function.

Theorem 5. *Let G be a k-safe state space. If $d \geq k$ and we run d-lookahead-RTA* on G with an admissible heuristic function h, it will never choose to move from a solvable state to an unsolvable state.*

Proof. Let s be the current state. Suppose s has an unsolvable child t. Since G is k-safe, it follows that $H(t) \leq k \leq d$. Since h^d is d-aware, it follows that $h^d(t) = \infty$, whence $f(t) = \infty$. If d-lookahead-RTA* moves from s to t, then it must be that $f(u) \geq f(t) = \infty$ for every child u of s, whence $h^d(u) = \infty$ for every child u of s. But since h^d is admissible, it follows that every child of s is unsolvable, whence s itself is unsolvable. □

Corollary 6. *Suppose G is k-safe and its initial state is solvable. If $d \geq k$ and we run d-lookahead-RTA* on G with an admissible heuristic function h, it is guaranteed to solve G.*

Proof. Let G' be the subgraph consisting of all solvable states of G. Since h^d is admissible, it follows from (Korf 1990) that RTA* using h^d (i.e., d-lookahead-RTA* using h) is guaranteed to solve G'. But from the above theorem it follows that d-lookahead-RTA*'s behavior in G will be identical to its behavior in G', hence d-lookahead-RTA* is also guaranteed to solve G. □

We say that a state s' is a *k-safe descendant (or child)* of s, if s' is a descendant (or a child) of s and the above corrollary holds for s'.

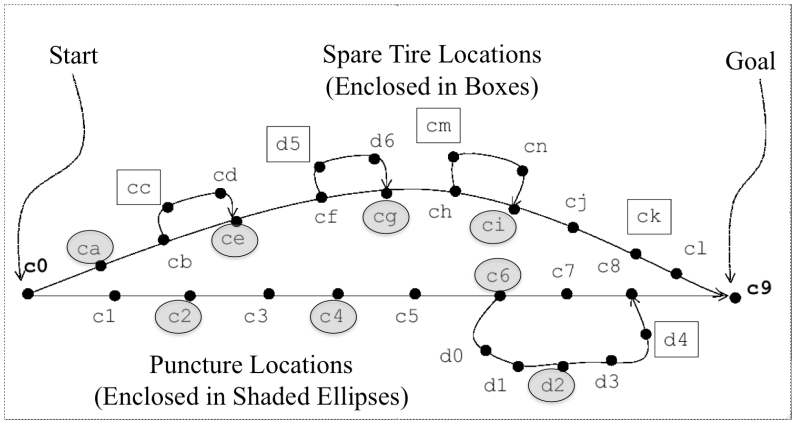

Figure 2: A Tireworld problem based on one in (Younes and Littman 2004). Unshaded boxes indicate tire-store locations, and shaded ellipses indicate puncture locations.

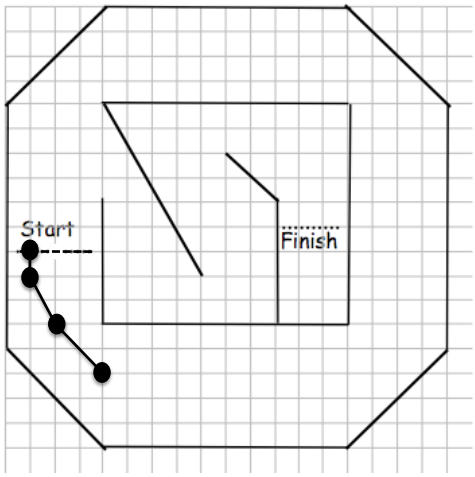

Figure 3: An illustration of the first few moves of one of the cars in a Racetrack game.

Analysis of Two Planning Domains

In this section, we analyze the deterministic versions of two well-known planning domains, Tireworld and Racetrack. We show that for any Tireworld or Racetrack map, it is easy to derive a value k such that all planning problems on that map are k-safe.

Tireworld

Tireworld was first introduced as a benchmark planning domain in the 2004 International Probabilistic Planning Competition (Littman and Younes 2004; Younes and Littman 2004). Our work does yet not deal with probabilistic planning (though we intend to do so in the future), so we now define a deterministic version of the domain.

In our deterministic version of Tireworld, there are n locations connected via bi-directional roads, forming an undirected graph whose nodes are the locations and whose edges are the roads (e.g., see Figure 2). Each road between two adjacent locations has the same fixed cost. Some of the locations on this graph are "tire-store locations" in which there are an unlimited number of spare tires. Other locations are "puncture" locations, at which the car will get a flat tire. There is a car in this domain that needs to go from an initial location to a final one. The car can carry at most one spare tire.

A *state* in a Tireworld problem is a triple $s = (\text{loc}(s), \text{flat}(s), \text{spare}(s))$, where $l(s)$ is the car's location, and $\text{flat}(s)$ and $\text{spare}(s)$ are booleans telling whether the car has a flat tire and whether it is carrying a spare. The car can move to an adjacent location iff $\text{flat}(s) = \text{F}$. If $s = (l, \text{F}, \text{F})$ and the car moves to a puncture location l', then the new state is $s' = (l', \text{T}, \text{F})$, i.e., the car now has a flat tire. If $s = (l, \text{F}, \text{T})$ and the car moves to a puncture location l', then the new state is $s' = (l', \text{F}, \text{F})$, i.e., the spare tire replaces the flat. If $s = (l, \text{F}, \text{F})$ or $s = (l, \text{F}, \text{T})$ and the car moves to a tire-store location l', then the new state is $s' = (l', \text{F}, \text{T})$, i.e., the car now has a spare tire if it didn't have one before.

Two tire-store locations are *neighbors* if the shortest (i.e., smallest number of edges) path between them contains no other tire-store locations. The following results establish conditions under which Tireworld is k-safe.

Lemma 7. *Let G be any solvable Tireworld state space in which the initial location is a tire-store location. Then every neighboring tire-store location that is reachable from s_0 is also solvable.*

Proof. Suppose G satisfies the conditions of the theorem, and let $s_0 = (l, \text{F}, \text{T})$ be the initial state. If l' is a neighboring tire-store location that is reachable from s_0, then there is a path π from s_0 to $s' = (l', \text{F}, \text{T})$. Between $l(s_0)$ and $l(s_i)$ there is at most one puncture location, for otherwise π would end at the second puncture location. Thus there is a path π' from s' back to $l(s_0)$. Thus since s_0 is solvable, so is s'. □

Lemma 8. *Let G be any solvable Tireworld state space in which the initial location is a tire-store location, and let k be the maximum distance between any two neighboring tire stores. Then G is k-safe.*

Proof. Let s be any state reachable from s_0, and let π be the path through which it is reached. Then the distance between s and the last-tire store l in π is $\leq k$, and it follows from Lemma 7 that (l, F, T) is solvable. Suppose $H(s) > k$. Then there is a simple path from s of length k. This path must contain a tire-store location l', and it follows from Lemma 7 that (l', F, T) is solvable. Since there is a path from s to (l', F, T), it follows that s is also solvable. □

Theorem 9. *Let G be any solvable Tireworld state space in which there is a puncture-free path from the initial location is a tire-store location, and let k be the maximum distance between any two neighboring tire stores. Then G is k-safe.*

Proof. Immediate from the lemma. □

Racetrack

Racetrack is a game that was popular during the late 1960s and early 1970s (Gardner 1986), and one of the authors of this paper remembers playing it around that time. The original Racetrack game was deterministic, but (Bonet and Geffner 2003) introduced a modified version of Racetrack using actions with nondeterministic outcomes. Below is a summary of the original deterministic version of the game, based on the description in (Wikipedia 2009).

To play Racetrack, the players need to draw a racetrack on graph paper. See Figure 3 for an example. Each player selects a symbol and marks a point on the start line. Players take turns moving. Each move has two parts: inertia and acceleration. Inertia just continues the speed and direction of the last turn. Acceleration then adds one square in any direction. If a player hits the wall, he/she loses.

If there are two or more players, the objective is to reach to the finish line from the start line before the other players do. If there is just one player, then the objective is to reach the finish line in the least number of turns (i.e., we can treat each edge as having the same fixed cost).

The following theorem establishes the k-safeness of the Racetrack planning domain:

Theorem 10. *Let G be a Racetrack problem on a grid of size $n \times n$. Let $k = 1/2 + \sqrt{1/4 + 2n}$. Then G is k-safe.*

Proof. A car's state in a Racetrack game can be described as a 4-tuple $s = (x, y, u, v)$, where (x, y) is the car's location and (u, v) is the car's velocity. The unsolvable states in a racetrack game are the ones where the car will inevitably hit the wall before it reaches a goal. The only way that the car can inevitably hit the wall is if it is going too fast to stop in time. The number of moves needed to stop the car will be $\max(|u|, |v|)$, where (u, v) is the car's velocity. On an $n \times n$ grid, the car's velocity is no more than (z, z), where $|z(z + 1)/2| \leq n$. From the quadratic formula, it follows that $z \leq 1/2 + \sqrt{1/4 + 2n}$. Hence it is possible to tell whether s is solvable by looking ahead k steps, where $k = 1/2 + \sqrt{1/4 + 2n}$. □

Related Work

In addition to the work cited in previous sections of this paper, several other approaches have been investigated to improve real-time heuristic search, and in particular, to reduce the heuristic computation time need to decide on the action to execute. One common approach has been the use offull-width limited-depth lookahead (Korf 1990; Shimbo and Ishida 2003; Furcy and Koenig 2000; Rayner et al. 2007), the use of search spaces generated by A*-like search algorithms (Koenig 2004; Koenig and Likhachev 2006).

RTA* has been extended to "learn" the heuristic values of states and actions as the search algorithm explores the underlying environment by planning and execution. The extended algorithm is called the Learning RTA* (LRTA*) as described in (Korf 1990). There have been several enhancement for LRTA* recently in order to generalize the learning component more robust to the uncertainty in hihgly dynamic environments, such as real-time strategy games. Examples of these generalizations include the use of priority queues (Rayner et al. 2007), dynamic lookahead detection and waypoint seelction during search (Bulitko et al. 2008; 2007b), and the use minimax game-tree search techniques (Koenig 2001b).

There have been other approaches developed for improving the performance of real-time heuristic search. Most notably perhaps is the use of *state abstraction* techniques as described in (Bulitko et al. 2007a). In a nutshell, state abstraction involves generating an abstract representation of the search space, which is much more condensed and has smaller number of abstract states to perform the search over, computing the heuristic functions in that abstract space, and using the heuristic information to select and execute actions in the actual state space. The abstract space is usually generated via clustering together the states that share a common property (e.g., graph-abstractions as in (Holte et al. 1996a; 1996b), or by generating a relaxation of the actual state space (eg., by ignoring the obstacles in a path-finding environment (Koenig, Tovey, and Smirnov 2003; Koenig 2004), or by perfoming a limited A*-like search in order to discover a sub-space of the actual search space in which the heuristic action selection can be done as mentioned above.

In summary, there are several techniques that can enable a search algorithm to compute the heuristic function faster by doing either a limited-depth lookahead or by searching all the way to the goal in a smaller space, in order to make decisions on which actions to execute faster. But to the best of our knowledge, these techniques do not make use of conditions such as our k-safeness property, in which all unsolvable states can be detected.

Conclusions and Future Work

As we discussed earlier, there are many real-time planning problems in which a bad choice of action can lead to an unsolvable part of the state space. We have described a a class of problems called k-safe problems, in which the consequence of such a choice become apparent within k moves after the choice is made. When k is not too large, this means that it is possible to detect and avoid unsolvable states in real time.

We have described k-lookahead-RTA*, a modified version of RTA* that works correctly in k-safe problems, by looking ahead far enough to see the consequences of bad choices. We have proved that k-lookahead-RTA* can solve any k-safe real-time planning problem. We have analyzed deterministic real-time versions of two well-known planning domains (Tireworld and Racetrack) in which the state space contain a large number of unsolvable states. Our analysis shows it is not hard to identify values of k for which the domains are k-safe.

Ongoing work. We have implemented k-lookahead-RTA* and have run experiments with it on both domains. Our experimental results, to be included in an upcoming paper, show that k-lookahead-RTA* can solve problems in these domains with a small per-move overhead.

Future work. We think it is likely that there are many real-time planning problems in which k-safeness conditions exist—not just deterministic problems such as the ones investigated in this paper, but also probabilistic planning problems. We think it will be straightforward to define a k-safeness condition for probabilistic real-time planning problems, and to modify k-lookahead-RTA* to work in such problems. This is one of our topics for future work.

Acknowledgments

This work was supported in part by AFOSR grant FA95500610405, NAVAIR contract N6133906C0149, DARPA's Transfer Learning and Integrated Learning Program, and NSF grant IIS0412812. The opinions in this paper are those of the authors and do not necessarily reflect the opinions of the funders.

References

Bonet, B., and Geffner, H. 2003. Labeled RTDP: Improving the Convergence of Real-Time Dynamic Programming. In *International Conference on Automated Planning and Scheduling (ICAPS-03)*, 12–21. AAAI Press.

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007a. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research* 30:51–100.

Bulitko, V.; Bjornsson, Y.; Lustrek, M.; Schaeffer, J.; and Sigmundarson, S. 2007b. Dynamic control in path-planning with real-time heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS-07)*.

Bulitko, V.; Lustrek, M.; Schaeffer, J.; Bjornsson, Y.; and Sigmundarson, S. 2008. Dynamic control in real-time heuristic search. *Journal of Artificial Intelligence Research* 32:419 – 452.

Erol, K.; Nau, D. S.; and Subrahmanian, V. S. 1995. Complexity, decidability and undecidability results for domain-independent planning. *Artificial Intelligence* 76(1–2):75–88.

Furcy, D., and Koenig, S. 2000. Speeding up the convergence of real-time search. In *AAAI-00*.

Gardner, M. 1986. Sim, chomp, and racetrack. In *Knotted Doughnuts and Other Mathematical Entertainments*. W.H. Freeman.

Gutmann, J.-S.; Fukuchi, M.; and Fujita, M. 2005. Real-time path planning for humanoid robot navigation. In *IJCAI*, 1232–1237.

Holte, R. C.; Mkadmi, T.; Zimmer, R. M.; and MacDonald, A. J. 1996a. Speeding up problem-solving by abstraction: A graph-oriented approach. *Artificial Intelligence* 85:321–361.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996b. Hierarchical hierarchical a*: Searching abstraction hierarchies efficiently. In *AAAI-96*, 530–535.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive a*. In *AAMAS*.

Koenig, S.; Tovey, C.; and Smirnov, Y. 2003. Performance bounds for planning in unknown terrain. *Artificial Intelligence* 147(1-2):253–279.

Koenig, S. 1999. Exploring unknown environments with real-time search or reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*.

Koenig, S. 2001a. Agent-centered search. *AI Magazine*.

Koenig, S. 2001b. Minimax real-time heuristic search. Technical Report GIT-COGSCI-2001/02, College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Koenig, S. 2004. A comparison of fast search methods for real-time situated agents. In *International Joint Conferences on Autonomous Agents and Multiagent Systems International Joint Conferences on Autonomous Agents and Multiagent Systems (AAMAS)*.

Korf, R. 1990. Real-time heuristic search. *Artificial Intelligence* 42(2-3):189–211.

Littman, M., and Younes, H. 2004. The First International Probabilistic Planning Competition (IPPC-04).

Rayner, D. C.; Davison, K.; Bulitko, V.; Anderson, K.; and Lu, J. 2007. Real-time heuristic search with a priority queue. In *IJCAI-07*.

Sherback, M.; Purwin, O.; and D'Andrea, R. 2006. Real-time motion planning and control in the 2005 cornell robocup system. In *Lecture Notes in Control and Information Sciences*, volume 335. Springer. 245–263.

Shimbo, M., and Ishida, T. 2003. Controlling the learning process of real-time heuristic search. *Artificial Intelligence* 146(1):1–41.

Stene, S. B. 2006. Artificial Intelligence Techniques in Real-Time Strategy Games - Architecture and Combat Behavior. *Institutt for datateknikk og informasjonsvitenskap*.

Weiss, B.; Naderhirn, M.; and del Re, L. 2006. Global real-time path planning for uavs in uncertain environment. In *Proc. IEEE Internat. Symp. on Intelligent Control*.

Wikipedia. 2009. Racetrack (game). `http://en.wikipedia.org/wiki/Racetrack_(game)`.

Younes, H., and Littman, M. 2004. Tire world domain. `http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume24/younes05a-html/node18.html`.