Escaping Heuristic Hollows in Real-Time Search without Learning

Carlos Hernández
Departamento de Ingeniería Informática
Universidad Católica de la Santísima Concepción
Concepción, Chile

Jorge A. Baier
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

***Abstract*—Real-time search is a standard approach to solving search problems in which agents have limited sensing capabilities and must act quickly. It is well known that real-time search algorithms like LRTA* and RTA* perform poorly in regions of the search space in which the heuristic function is very imprecise. Approaches that use lookahead or learning are used to overcome this drawback. They perform more computation in the planning phase compared to LRTA* and RTA*. In this paper we propose Path Real-Time A* (PRTA*), an algorithm that, like LRTA*, performs little computation in the planning phase, but that, unlike LRTA*, terminates even if the problem does not have a solution. We show that our algorithm outperforms LRTA* and RTA* in standard real-time benchmark problems. Furthermore, we show that in some cases, PRTA* may also outperform lookahead- or learning-enabled algorithms but carrying out significantly less computation.**

I. Introduction

Autonomous agents acting in dynamic, previously unknown domains, often have to make decisions quickly. Real-time (e.g., [1]) or agent-centered search [2] is the standard paradigm for solving search problems in those settings. Instead of building a conditional plan at the outset, real-time algorithms interleave planning and execution. As such, they usually run a computationally inexpensive sense-plan-act cycle, in which the environment is observed, an action is selected, and then executed. The cycle is executed until a solution is found. Just as in standard A* search [3], a heuristic function is used to guide action selection.

LRTA* and RTA* [4] are examples of real-time search algorithms. Their planning phase takes little time; indeed, bounded by a constant. Nonetheless, such a reduced computational effort comes at a price: in problems with heuristic hollows or depressions, they perform poorly [5]. Intuitively, a heuristic hollow is a closed region of the search space in which the heuristic is unrealistic with respect to the heuristic value of the states in the border of the region. In problems with hollows, LRTA* and RTA* usually become "trapped", and sometimes re-visit several states before exiting the hollow.

Approaches to overcome this pitfall exist and can be grouped in two categories. First, algorithms that do additional lookahead (e.g., [6], [7], [8], [9]), which provide a more elaborate action selection mechanism. Second, algorithms that do extended learning (e.g., [10], [11], [12]), which involves updating the heuristic value of several states in one cycle. Compared to LRTA* and RTA*, these approaches perform more extensive computation in the planning phase but generally perform better in presence of heuristic hollows.

In this paper we propose Path Real-Time A* (PRTA*), a real-time search algorithm that, like LRTA*, requires little computation in the planning phase, but that, unlike LRTA*, requires fewer moves to exit heuristic hollows. The key idea in PRTA* is to avoid re-visiting states identified as part of a heuristic hollow. We show that PRTA* outperforms LRTA* and RTA* in standard benchmarks, and that in some cases it outperforms algorithms that perform much more computation in the planning phase. In addition PRTA* has interesting theoretical properties: it is sound and complete even when the problem has no solution. To our knowledge, this property is not enjoyed by any existing real-time algorithm.

The paper is organized as follows. We explain the basic concepts of real-time search in the next section. We continue explaining the concept of heuristic hollow in more detail. We then describe our algorithm and continue with a theoretical and experimental analysis. Finally, we briefly sketch our conclusions.

II. Preliminaries

We carry out our search in undirected search spaces with positive-cost actions. A search problem is defined by the tuple $P = (X, A, c, s_0, G)$, where (X, A) is a digraph that represents the search space. The set X represents the *states* and the arcs in A represent all available actions. We assume A does not contain elements of form (x, x). In addition, the cost function $c : A \mapsto \mathbb{R}^+$ associates a positive cost to each of the available actions. Finally, $s_0 \in X$ is the start state, and $G \subseteq X$ is a set of goal states. We say that a search space is *undirected* if whenever (u, v) is in A then so is (v, u). Unless specified otherwise, we assume that in undirected spaces $c(u, v) = c(v, u)$, for all $(u, v) \in A$. We define $k(u, v)$ as a function that returns the cost of the minimum-cost path between states u and v. The successors of a state u are defined by $Succ(u) = \{v \mid (u, v) \in A\}$. A heuristic function $h : X \mapsto [0, \infty)$ associates to each state x an approximation $h(x)$ of the cost of a path from x to a goal state g. The

minimum cost incurred by achieving a goal state from x is denoted by $h^*(x)$. We say that h is *admissible* iff $\forall x \in X, h(x) \leq h^*(x)$. h is *consistent* iff $0 \leq h(x) \leq c(x,w) + h(w)$ for all states $w \in Succ(x)$. If $s' \in Succ(s)$ we say that s and s' are *neighbors*.

A. Real-Time Search

The objective of a real-time search algorithm is to make an agent travel from an initial state to a goal state performing, between moves, an amount of computation bounded by a constant. An example real-time situation is the one we discuss throughout the rest of the paper: an agent moving in a grid-like environment in which cells may be blocked or unblocked. The agent has a memory capable of storing its current belief about the structure of the search space, which it initially regards as obstacle-free (this is usually referred to as the *free-space assumption* [13]). The agent is capable of a limited form of sensing: only obstacles in the neighbor cells can be detected. When obstacles are detected, the agent updates its memory map accordingly.

```
1 s ← s_0
2 while s ∉ G do
3 |   for each w ∈ Succ(s) do update c(s, w)
4 |   y ← argmin_{w∈Succ(s)} c(s, w) + h(w)
5 |   h(s) ← max{h(s), c(s, y) + h(y)}
6 |   Move the agent to y and do s ← y
7 end
```

Figure 1. Pseudo code for LRTA*.

Learning Real-Time A* (LRTA*) [4] (Fig. 1) is an algorithm that solves real-time search problems. It iteratively executes an observe-lookahead-update-act cycle until the goal is reached. The procedure has three local variables. Variable s stores the current position of the agent. Variable $c(s, s')$ contains the cost of moving from state s to a successor s'. Variable h is such that $h(s)$ contains the heuristic value for the state. All three variables may change over time. Initially c is such that no obstacles are assumed; i.e., $c(s, s') < \infty$ for any two neighbor states s, s'. The initial value of h is given as a parameter.

In the observation phase (Line 8), LRTA* updates the cost to reach each of the neighbors of the current position; this essentially involves setting $c(s, w)$ to infinity if a successor w of s is blocked. In the lookahead phase (Line 4), it determines where to proceed next based on the heuristic value of the neighbor states and the cost to reach them. In the update phase (Line 5), it updates (i.e., *learns*) the heuristic function of the current state based on the heuristics of its successors and the cost of reaching them. The update is intuitively correct in terms of a consistency criterion: if h is consistent, then the value of h is changed to the highest possible value that would allow $h(s)$ to remain consistent. This update is also a limited form of learning (hence the name of the algorithm). Finally, in the acting phase (Line 6) it changes the position of the agent to the best selected state.

LRTA* has interesting properties. If the initial heuristic is consistent, then it remains consistent throughout execution. If a solution exists, then its learning mechanism guarantees that such solution will be found [4]. However, if a solution does not exist, the algorithm enters into an infinite loop.

III. Heuristic Hollows

LRTA* performs poorly when it visits a state in a *heuristic hollow*. Intuitively, a heuristic hollow is a bounded region of the search space containing states whose heuristic value is unrealistically low with respect to the heuristic values in the border of the hollow. More precisely, a hollow is a connected component of states completely surrounded by a set of states we call *border*. For any state s in the hollow there is a state s' in the border such that $h(s)$ is lower than the sum of $h(s')$ and the optimal cost of reaching s' from s. Our definition of hollow is similar to Bulitko's notion of γ-traps [14], and is related to the concept of *heuristic depressions* [5]; both of these authors discuss the (poor) performance of LRTA* in similar circumstances.

The upper left grid in Fig. 2 sketches a real-time search problem in which we assume that the cost of moving to an adjacent cell is 1. The shaded region shows a heuristic hollow in which the heuristic value of all its search nodes are unrealistic underestimations with respect to the border (cell C6). For example, the heuristic value for state A3 (2) is lower than the sum of the heuristic value of state C6 (7) and the cost of reaching state C6 (5) from A3.

It is known that LRTA* behaves poorly in presence of heuristic hollows [5]. To see this, assume that LRTA* visits a state in a hollow and that the solution node lies outside the hollow. To exit the hollow the agent must follow a path in its interior, say, $s_1 \ldots s_n$, finally choosing a state in the border of the region, say s_e. While visiting s_n the agent chooses s_e as the next move, which means that s_e *minimizes* the estimated cost to reach a solution among all the neighbors of s_n. In problems with uniform cost, this can only happen if $h(s_e)$ is *lower or equal* than the heuristic value of all other neighbors of s_n. This can only happen if the heuristic values of all the neighbors of s inside the hollow have been updated (increased) in previous iterations. For LRTA*, the update process may be quite costly: in the worst case, indeed, *all* states in the depression may need to be updated and each state may need to be updated several times.

Fig. 2 shows 6 agent moves of an execution of LRTA* in our example. It can be observed that some states are visited several times. LRTA* actually requires more than 20 agent moves to exit the hollow, visiting the same states several times.

There exist two main approaches that aim at reducing the number of moves required to exit from heuristic hollows.

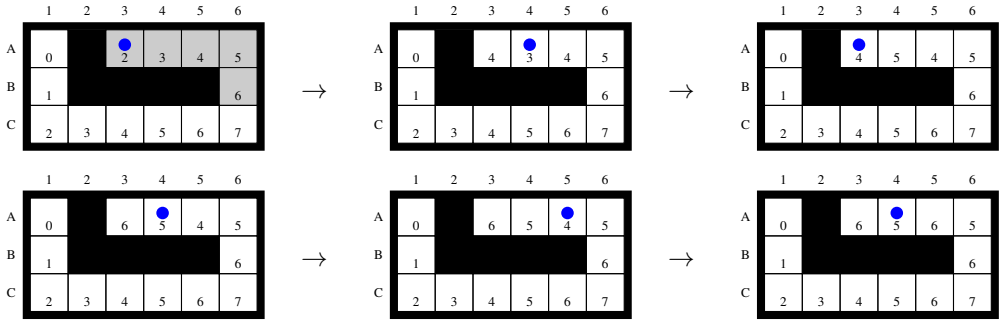

Figure 2. First 6 agent moves of LRTA* in our example problem: a 4-connected grid world defining an undirected space with uniform action costs equal to 1, where the initial state is A3, and the goal state is A1. The number in each cell denotes the heuristic value for the corresponding state. The heuristic used is the *manhattan distance*. We assume ties are broken by choosing first left then top then right then bottom adjacent cell. The position of the agent is given by the dot.

The first and most common approach is to make a greater lookahead. Essentially, this approach involves carrying out a limited search before deciding where to move the agent. LSS-LRTA* [6], LRTS [7], RTAA* [8] and TBA* [9] are examples of lookahead algorithms. Another approach is to make a larger number of updates in the update phase. LRTA* (k) [10], P-LRTA* [11] and LRTA*$_{\text{LS}}(k)$ [12] are examples such algorithms. We call these algorithms *learning algorithms*. Some algorithms (e.g. LSS-LRTA*) combine lookahead and learning.

Lookahead and learning algorithms make a greater effort in terms of number of state expansions than LRTA* before the acting phase. Those algorithms can solve a search problem with fewer agent moves than LRTA*, but their techniques can be rendered infeasible in presence of very tight real-time constraints.

IV. Path Real-Time A*

We have seen above that a major issue in solving real-time search problems is the presence of heuristic hollows. This issue is addressed by approaches that perform sometimes significant *additional computation* before the acting phase. This section presents PRTA*, a real-time search algorithm that allows escaping from hollows quickly performing *little*, *constant-time* computation before the acting phase.

There are two main conceptual differences between PRTA* and LRTA*. The first difference is that once a state is identified as being part of a heuristic hollow, it is marked as not *visitable*; this means that the agent will not visit that state in the future. The second main difference is that PRTA* *does not* update the heuristic of the state, but rather provides a mechanism to avoid entering infinite loops (recall that learning in LRTA* is a mechanism that guarantees termination). Specifically, PRTA* stores the sequence of states visited by the agent in a *path* variable implemented as a stack. If a state with no visitable successors is reached, the agent goes back to its previously visited state.

Fig. 3 shows the pseudo code for PRTA*. For any state

```
1  function PRTA* (P)
2      s ← s_0
3      path.PUSH(s)
4      seen ← {}
5      for each s ∈ X do s.updaterule ← false
6      while s ∉ G do
7          s ← seen ∪ {s}
8          for each w ∈ Succ(s) do update c(s, w)
9          N ← {w ∈ succ(s) | w.updaterule = false}
10         y ← argmin_{w∈N} c(s, w) + h(w)
11         if y ≠ Null then
12             if h(s) < c(s, y) + h(y) then
13                 s.updaterule ← true
14             end
15             Move the agent to y
16             s ← y
17             path.PUSH(s)
18         else
19             s.updaterule ← true
20             path.POP()
21             if path = EmptyStack then
22                 return no-solution
23             end
24             Move the agent to path.TOP()
25             s ← path.TOP()
26         end
27         return solution-found
28     end
29 end
```

Figure 3. Pseudo-code for Path Real-Time A*.

s, the boolean $s.updaterule$ may become true if s is determined to be in a hollow. Hence $s.updaterule$ is true when a state is not visitable. Initially $s.updaterule$ is set to false for all states (Line 5). Immediately after a move (Lines 2 and 16), the state the agent moves to is added to the *path*. The algorithm selects the next node to visit using the same rule as LRTA*, but only states not in a hollow are considered. If the current state is in a hollow the algorithm marks it in Line 13. Here we determine that an agent is in a hollow by considering the set of successors as the border of the hollow. Finally, when no valid successors for a node exist, the agent returns to the previously visited position, marking the current state to avoid re-visiting it (Lines 19–25).

The variable $seen$ contains all states that have been visited. It is only needed to simplify the proofs that come in the next section, and thus it does not need to be included in an implementation of PRTA*.

Fig. 4 shows the execution of PRTA* on the same example problem of Fig. 2. Six agent moves suffice to exit the hollow in this situation.

V. Evaluation

A. Theoretical Analysis

We first prove that PRTA* always terminates, independent of whether or not P has a solution. Many real-time search algorithms (e.g. LRTA*, LSS-LRTA*, RTA*) do not have such a property.

Lemma 1 *PRTA* terminates on any search problem P.*

Proof: We proceed by contradiction, assuming the algorithm executes forever. In an infinite execution, a state is visited by the algorithm infinitely often. This can happen in two (non-exclusive) cases: (1) because a state is added and removed infinitely often from $path$, or (2) because the algorithm traverses a path that contains infinite copies of a state. Case (1) is not possible since Line 19 prevents a state from being selected in the future when it has been popped from the stack. In case (2), variable $path$ is equal to $\sigma'\sigma\sigma\cdots$, were σ' and σ are finite sequences of states. Let $\sigma = s_1 s_2 \cdots s_n$. Given the condition in Line 9, we have that $s_i.updaterule = false$ for any s_i in σ; otherwise, the algorithm could have not moved to s_i infinitely often. Thus we have that $h(s_i) \geq c(s_i, s_{i+1}) + h(s_{i+1})$, for every $i \in [1, n]$, since the condition in Line 12 should evaluate to false for every s_i in σ. Summing up the inequalities, we obtain $h(s_1) \geq K(s_1, s_n) + h(s_n)$, where $K(s_1, s_n)$ represents the (positive) added cost of moving between s_1 and s_n. In particular, this means that

$$h(s_n) < h(s_1). \tag{1}$$

When the algorithm has just moved to s_n ($s = s_n$), the best successor y of s_n is s_1 (because of our assumption that σ is visited infinitely often). Here, the condition in Line 12 evaluates to true because of Inequality 1, and thus $s_n.updaterule$ is set to $true$ in Line 13, which contradicts our initial assumption, finishing the proof. ■

Theorem 1 *PRTA* is sound and complete.*

Proof: We prove (1) that if the algorithm returns that a solution exists, then there is a path from s to G in the input problem, and (2) that if the algorithm returns "no solution", then no solution exists in P. (1) is trivial because if the algorithm reaches Line 6, it has followed a path to a goal state.

For (2) we prove the equivalent statement: if the algorithm returns in Line 20, then variable $seen$ contains all search states in the search space reachable from s_0.

Assume (2) is false. Then after returning $seen$ does not contain a state reachable from s_0. Since $seen$ only contains states that are reachable from s_0, after returning in Line 22, there is a state t that is the successor of a state $r \in seen$ such that $t \notin seen$. Since $r \in seen$, r must have been in the stack $path$ at some point during the execution (note that every time the agent moves to a position, such a position is added to the stack). Furthermore, because the algorithm returns in Line 22, r was eventually popped from $path$. When r is popped in Line 20, y must have been equal to $Null$, which means that either r has no successors (which is not true by our assumption) or $w.updatereule = true$ for all $w \in succ(r)$. In particular $t.updaterule = true$, and thus at some point the algorithm executed Line 13 for $s = t$. Hence t was added to $seen$ in Line 6, which contradicts our assumption. Since by Lemma 1, the algorithm always terminates, we conclude that PRTA* is complete.

■

B. Experimental Evaluation

We compared the performance of PRTA*, LRTA* and RTA* [4] at solving real-time goal-directed navigation problems in initially unknown terrain. The three algorithms make only one cell expansion in the lookahead phase. For fairness, we use comparable implementations.

We assume the agent operates on undirected eight-neighbor grids. Horizontal and vertical movements have action cost 1, whereas diagonal movements have cost $\sqrt{2}$. The agent does not know in advance which cells are blocked. It always observes which unblocked cell it is in, observes the blockage status of its eight neighboring cells and can then move to any of the unblocked neighbor cells.

Since the existence of heuristic hollows depends on the quality of the user-given heuristic (indeed, hollows are more frequent in bad heuristics), we studied the performance of our algorithm with heuristics of different quality. We used three user-given heuristics: the octile distance [15]—a high-quality heuristic for these problems—, the maximum

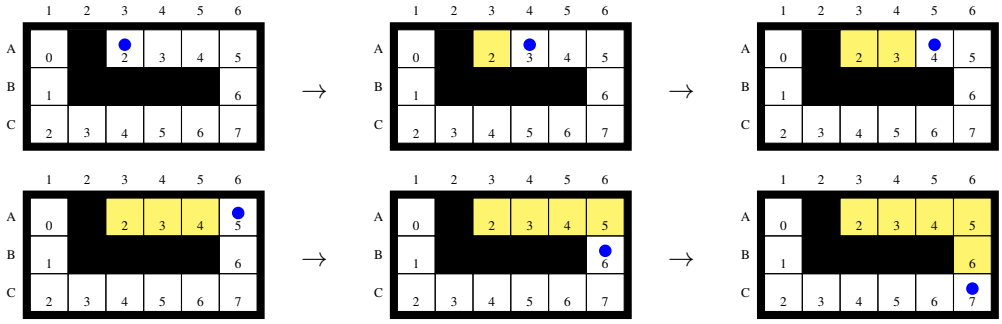

Figure 4. First 6 agent moves of PRTA* in our example problem. Shaded areas correspond to states marked as in a hollow.

of the absolute differences of the x and y coordinates of two cells—a medium-quality heuristic—, and the minimum of the absolute differences of the x and y coordinates of two cells—a heuristic known to perform poorly on these problems.

We use four computer game maps adapted from the game *World of Warcraft*. The game map 1 has 169×169 cells, the game map 2 has 134×137 cells, and the game maps 2 and 3 have 128×128 cells. Fig. 5 shows the four game maps.

For each test case in game maps, we choose the start cell randomly from the cells whose x-coordinates range from 1 to 20 and the goal cell randomly from the cells whose x-coordinates ranges from 150 to 169 in game map 1, from 115 to 134 in game map 2, and from 109 to 128 in game map 3 and 4. In order to make the game map environment navigation problem as difficult as possible, we choose the start and goal cells from the left and right sides of the map, respectively, to ensure that they are far apart. We average our experimental results over 4000 test cases for the game maps (1000 test cases for each particular game map).

Fig. 6 shows a measure for solution quality: the average solution cost per test case. Furthermore, it shows two measures for the efficiency of the search: the average number of expanded cells (per test case), and the total runtime (per test case) in milliseconds. All experiments were run on a Linux PC with a Pentium QuadCore 2.33 GHz CPU and 8 GB RAM.

As shown in Fig. 6, PRTA* yields a smaller solution cost, a smaller average number of cells expansions and smaller total runtime than LRTA* and RTA* for the three heuristic qualities. In addition, Fig. 6 shows that PRTA* is more reliable than LRTA* and RTA* in presence of bad heuristics. In fact the solution cost of LRTA* with the medium-quality heuristic is similar to solution cost of PRTA* with the low-quality heuristic, and PRTA* with the medium-quality heuristic yields a smaller solution cost, a smaller number of cells expansions and smaller total runtime per test case than LRTA* and RTA* used along with the high-quality heuristic.

We compared PRTA* with RTAA* and LRTA$^*_{\text{LS}}(k)$, a lookahead and learning algorithm respectively, using the high-quality heuristic. Fig. 7 reports the results. RTAA* and LRTA*(k) use a parameter to control the amount of lookahead and learning respectively. When the lookahead parameter is increased, the quality of the solutions typically improve; nonetheless, more nodes are typically expanded, yielding a less efficient search. We set the parameter of RTAA* and LRTA$^*_{\text{LS}}(k)$ at 10. For that value, the solution quality obtained by RTAA* and LRTA$^*_{\text{LS}}(k)$ are similar to that obtained by PRTA*. With lower values for the parameter, the solution quality of PRTA* is better than that of RTAA* and LRTA$^*_{\text{LS}}(k)$. The solution cost of the three algorithms are similar, but PRTA* is more efficient. To obtain a similar solution cost RTAA*(10) expands 4.6 times more cells and is 4.5 times slower than PRTA*. On the other hand, LRTA$^*_{\text{LS}}(10)$ expands 4.4 times more cells and is 6.0 times slower than PRTA*.

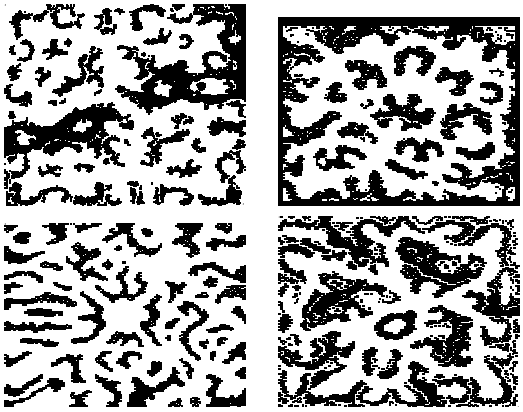

Figure 5. The four game maps used for experiments.

VI. Summary and Conclusions

We have presented PRTA*, a real-time search algorithm that explicitly avoids visiting states in a heuristic hollow. Our experiments on game benchmarks show that PRTA* substantially outperforms LRTA* and RTA*, which are algorithms that carry out a similar amount of computation in the planning phase. PRTA* also may outperform learning and lookahead algorithms which perform much more extensive

Heuristic Quality	Solution Cost		
	LRTA*	RTA*	PRTA*
high	1,958	1,608	**599**
medium	3,181	2,700	**830**
low	8,093	5,778	**3,199**

(a)

Heuristic Quality	Expansions per Test Case		
	LRTA*	RTA*	PRTA*
high	1,753	1,369	**526**
medium	2,999	2,330	**774**
low	7,736	5,105	**3,027**

(b)

Heuristic Quality	Time per Test Case		
	LRTA*	RTA*	PRTA*
high	0.23	0.21	**0.13**
medium	0.33	0.31	**0.17**
low	0.76	0.66	**0.48**

(c)

Figure 6. PRTA* performance comparison against algorithms that make only one cell expansion before the acting phase. (a) shows average solution cost, (b) shows average number of expansions, and (c) shows time per test case.

	Cost	Expansions	Time
PRTA*	599	526	0.13
RTAA*(10)	613	2,399	0.58
LRTA$^*_{\text{LS}}(10)$	527	2,318	0.78

Figure 7. PRTA* compared with algorithms that make more than one cell expansion before the acting phase.

computation in the planning phase. PRTA* is also sound and complete, even if the problem does not have a solution; a property that does not hold true for other well-known real-time search algorithms. Our results suggest that learning—the prevalent technique in real-time search—may not be the best choice for real-time search algorithms. PRTA* can be naturally extended to do more lookahead both for selecting a best move and for marking states, and thus we expect future extensions to be competitive or superior to learning and lookahead algorithms.

ACKNOWLEDGMENTS

We are grateful to Nathan Sturtevant for providing us with the game maps. Carlos Hernández was partly funded by Fondecyt project #11080063. Jorge Baier is grateful to Pontificia Universidad Católica de Chile for the funding provided through its VRI grant number 38/2010.

REFERENCES

[1] G. Weiss, Ed., *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. Cambridge, MA: MIT Press, 1999.

[2] S. Koenig, "Agent-centered search," *Artificial Intelligence Magazine*, vol. 22, no. 4, pp. 109–131, 2001.

[3] P. E. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimal cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, 1968.

[4] R. E. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2-3, pp. 189–211, 1990.

[5] T. Ishida, "Moving target search with intelligence," in *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI)*, 1992, pp. 525–532.

[6] S. Koenig, "A comparison of fast search methods for real-time situated agents," in *Proceedings of the 3rd International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, 2004, pp. 864–871.

[7] V. Bulitko and G. Lee, "Learning in real time search: a unifying framework," *Journal of Artificial Intelligence Research*, vol. 25, pp. 119–157, 2006.

[8] S. Koenig and M. Likhachev, "Real-time adaptive A*," in *Proceedings of the 5th International Joint Conference on Autonomous Agents and Multi Agent Systems (AAMAS)*, 2006, pp. 281–288.

[9] Y. Björnsson, V. Bulitko, and N. R. Sturtevant, "TBA*: Time-bounded A*," in *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 431–436.

[10] C. Hernandez and P. Meseguer, "LRTA*(k)," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, 2005, pp. 1238–1243.

[11] D. C. Rayner, K. Davison, V. Bulitko, K. Anderson, and J. Lu, "Real-time heuristic search with a priority queue," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2007, pp. 2372–2377.

[12] C. Hernandez and P. Meseguer, "Improving LRTA*(k)," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, 2007, pp. 2312–2317.

[13] S. Koenig, C. A. Tovey, and Y. V. Smirnov, "Performance bounds for planning in unknown terrain," *Artificial Intelligence*, vol. 147, no. 1-2, pp. 253–279, 2003.

[14] V. Bulitko, "Learning for adaptive real-time search," *Computing Research Repository*, vol. cs.AI/0407016, 2004. [Online]. Available: http://arxiv.org/abs/cs.AI/0407016

[15] N. R. Sturtevant and M. Buro, "Partial pathfinding using map abstraction and refinement," in *Proceedings of the 20th National Conference on Artificial Intelligence (AAAI)*, 2005, pp. 1392–1397.