Real-Time Heuristic Search with a Priority Queue

D. Chris Rayner, Katherine Davison, Vadim Bulitko, Kenneth Anderson, Jieshan Lu
University of Alberta
Department of Computing Science
Edmonton, Alberta, Canada T6G 2E8
{rayner|kdavison|bulitko|anderson|jieshan}@ualberta.ca

Abstract

Learning real-time search, which interleaves planning and acting, allows agents to learn from multiple trials and respond quickly. Such algorithms require no prior knowledge of the environment and can be deployed without pre-processing. We introduce Prioritized-LRTA* (P-LRTA*), a learning real-time search algorithm based on Prioritized Sweeping. P-LRTA* focuses learning on important areas of the search space, where the importance of a state is determined by the magnitude of the updates made to neighboring states. Empirical tests on path-planning in commercial game maps show a substantial learning speed-up over state-of-the-art real-time search algorithms.

1 Introduction

We introduce Prioritized Learning Real-Time A* (P-LRTA*). P-LRTA* prioritizes learning to achieve significant speed-ups over competing real-time search algorithms.

Real-time search algorithms interleave planning and acting to generate actions in user-bounded time. These algorithms are also *agent-centered* [Koenig, 2001]: they do not require *a priori* knowledge of the map, they learn the transition model by interacting with the environment, and they improve their solutions by refining a heuristic function over multiple trials. Algorithms with these three properties can be used for path-planning in computer games, virtual reality trainers [Dini *et al.*, 2006], and robotics [Koenig and Simmons, 1998]. In each of these domains, agents plan and act in potentially unknown environments; when identical tasks must be solved in succession, the agent has an opportunity to improve its performance by learning. Examples include commute-type tasks, such as resource collection and patrolling.

Our focus is on the learning process. Since learning takes place on-line and on-site, it is critical to minimize learning time by converging rapidly to high-quality solutions.

We build on existing research and present a learning real-time search algorithm that makes learning more experience-efficient by prioritizing updates [Moore and Atkeson, 1993]. States that are deemed important are updated before other states, where importance is determined by the magnitude of the updates made to a neighboring state. Prioritizing heuristic updates for efficient learning is motivated by two observations about asynchronous dynamic programming [Barto *et al.*, 1995]. First, since states are updated one at a time, new state values depend upon the order in which the updates occur; a judicious update ordering can make individual updates more effective. Second, a significant update to one state often affects its neighbors, and giving priority to these neighbors can focus computation on the most worthwhile updates.

The rest of the paper is organized as follows: first, we formally describe the problem. Next, we examine and identify the shortcomings of related algorithms, and motivate our approach. We then provide a comprehensive description of our novel algorithm, and its general properties. Next we justify our performance metrics and conduct an empirical evaluation. We conclude by considering possible extensions to P-LRTA*.

2 Problem Formulation

We focus on problems defined by the tuple (S, A, c, s_0, s_g, h_0): S is a finite set of states, A is a set of deterministic actions that cause state transitions, and $c(s, a)$ is the cost of performing action $a \in A$ in state $s \in S$; if executing action a in state s returns the agent to state s, that action is said to be *blocked*. Blocked actions are discovered when they are within the agent's *visibility radius*, the distance at which the agent can sense the environment and update its transition model.

The agent begins in a particular start state $s_0 \in S$ and aims to reach the goal state s_g. Upon reaching a goal state, the agent is teleported back to the start state and commences a new trial. A learning agent has converged when it completes a trial without updating the heuristic value of any state.

For every action executed, the agent incurs a cost associated with that action. In general, any real-valued costs lower-bounded by a positive constant can be used. We assume that the state space is safely explorable insomuch as a goal state is reachable from any state.

The agent deals with its initial uncertainty about whether an action is blocked in a particular state by assuming that all actions from all states are unblocked; this belief is known as the *freespace assumption* [Koenig *et al.*, 2003]. As the agent explores, it can learn and remember which actions are possible in which states, and plan accordingly.

3 Related Work

A real-time agent is situated at any given time in a single state known as its *current state*. The current state can be changed by taking actions and incurring an execution cost. An agent can reach the goal from the current state in one of two ways: the agent can plan a sequence of actions leading to the goal and then act, or the agent can plan incompletely, execute this partial plan, and then repeat until it reaches the goal. We review search algorithms for both techniques.

An agent using Local Repair A* (LRA*) [Silver, 2005] plans a complete path from its current state to the goal, and then executes this plan. This path is optimal given the agent's current knowledge. The agent may discover that its planned path is blocked, and at this point will stop and re-plan with updated world knowledge. Each plan is generated with an A* search [Hart *et al.*, 1968], so the time needed for the first move and for re-planning is $O(n \log n)$, where n is the number of states. Algorithms such as Dynamic A* (D*) [Stentz, 1995] and D* Lite [Koenig and Likhachev, 2002] efficiently correct the current plan rather than reproduce it, but they do not reduce the computation needed before the first move of each trial can be executed. A large first-move delay negatively affects an agent's responsiveness in interactive environments such as computer games and high-speed robotics.

In its simplest form, Korf's Learning Real-Time A* (LRTA*) [1990] updates the current state only with respect to its immediate neighbors. We refer to this version of LRTA* as LRTA*(d=1). If the initial heuristic is non-overestimating then LRTA* converges optimally [Korf, 1990]. The amount of travel on each trial during learning can oscillate unpredictably, causing seemingly irrational behavior. Heuristic weighting and learning a separate upper bound reduces the path length instability [Shimbo and Ishida, 2003; Bulitko and Lee, 2006], but can result in suboptimal solutions.

We define the *local search space* (LSS) as those states whose neighbors are generated when planning a move. LRTA*(d=1) only looks at the current state's immediate neighbors, but a deeper lookahead gives an agent more information to decide on the next action [Korf, 1990]. For instance, LRTS [Bulitko and Lee, 2006] considers all of the states within a radius d of the current state. Alternatively, Koenig uses an LSS defined by a partial A* search from the current state to the goal [2004; 2006].

Updating more than one state value in each planning stage can result in more efficient learning. One example of this is physical backtracking, which helps to keep state updates in close physical proximity to the agent: SLA*, SLA*T [Shue and Zamani, 1993a; 1993b; Shue *et al.*, 2001], γ-Trap [Bulitko, 2004], and LRTS [Bulitko and Lee, 2006] all physically return to previously visited states, potentially reducing the number of trials needed for convergence with the possibility of increasing travel cost. Alternatively, LRTA*(k) [Hernández and Meseguer, 2005b; 2005a] uses *mental* backups to decrease the number of convergence trials. These two techniques are difficult to combine, and a recent study demonstrates the fragile and highly context-specific effects of combining physical and mental backups [Sigmundarson and Björnsson, 2006].

PRIORITIZED-LRTA*(s)

while $s \neq s_g$ **do**
  STATEUPDATE(s)
  repeat
    $p = queue.pop()$
    if $p \neq s_g$ **then**
      STATEUPDATE(p)
    end if
  until N states are updated or $queue = \emptyset$
  $s \Leftarrow$ neighbor s' with lowest $f(s, s') = c(s, s') + h(s')$
end while

Figure 1: The P-LRTA* agent updates the current state, s, and as many as N states taken from the queue, where N is an input parameter defined by the user. The agent takes a single greedy move, and then repeats this process until reaching the goal state, s_g.

Koenig's LRTA* [Koenig, 2004] updates the heuristic values of all LSS states on each move to accelerate the convergence process. While the Dijkstra-style relaxation procedure it uses produces highly informed heuristics, the process is expensive, and limits the size of LSS that can be used in real-time domains [Koenig and Likhachev, 2006].

In an attempt to reduce convergence execution cost, PR-LRTS [Bulitko *et al.*, 2005] builds a hierarchy of levels of abstraction and runs search algorithms on each level. The search at the higher levels of abstraction constrains lower-level searches to promising sections of the map, reducing exploration in lower levels. As a result, convergence travel and first move delay are improved at the cost of a complex implementation and the additional computation required to maintain the abstraction hierarchy during exploration.

A different line of research considers sophisticated update schemes for real-time dynamic programming algorithms. As Barto, Bradtke, and Singh note, "[the] subset of states whose costs are backed up changes from stage to stage, and the choice of these subsets determines the precise nature of the algorithm" [1995]. Prioritized Sweeping, a reinforcement learning algorithm, performs updates in order of priority [Moore and Atkeson, 1993]. A state has high priority if it has a large potential change in its value function. Only states with a potential update greater than $\epsilon (\epsilon > 0, \epsilon \in \mathbb{R})$ are added to the queue. Prioritized Sweeping is shown to be more experience efficient than Q-learning and Dyna-PI [Moore and Atkeson, 1993], and is a core influence for our algorithm.

4 Novel Algorithm

Prioritized-LRTA* (P-LRTA*) combines the ranked updates of Prioritized Sweeping with LRTA*'s real-time search assumptions of a deterministic environment and a non-trivial initial heuristic. In this section we describe P-LRTA*'s algorithmic details, then comment on the nature of its execution.

A P-LRTA* agent has a planning phase and an acting phase which are interleaved until the agent reaches the goal (Figure 1). During its planning phase, the agent gains knowl-

STATEUPDATE(s)

find neighbor s' with the lowest $f(s, s') = c(s, s') + h(s')$
$\Delta \Leftarrow f(s, s') - h(s)$
if $\Delta > 0$ **then**
  $h(s) \Leftarrow f(s, s')$
  for all neighbors n of s **do**
    ADDTOQUEUE(n, Δ)
  end for
end if

Figure 2: The value of state s is set to the lowest available $c(s, s') + h(s')$, where $c(s, s')$ is the cost of traveling to s', and $h(s')$ estimates the distance from s' to the goal. If the value of s changes, its neighbors are enqueued.

ADDTOQUEUE(s, Δ_s)

if $s \notin queue$ **then**
  if $queueFull()$ **then**
    find state $r \in queue$ with smallest Δ_r
    if $\Delta_r < \Delta_s$ **then**
      $queueRemove(r)$
      $queueInsert(s, \Delta_s)$
    end if
  else
    $queueInsert(s, \Delta_s)$
  end if
end if

Figure 3: State s is inserted into the queue if there is room in the queue or if its priority is greater than that of some previously enqueued state r. If a state has already been enqueued, it will not be inserted a second time into the queue.

edge about how to navigate the world by updating the distance heuristics at select states. During its acting phase, the agent simply uses greedy action-selection to choose it next move. We describe these two phases in detail.

At the beginning of the planning phase, the agent updates the current state s by considering its immediate neighbors (Figure 2). The amount by which the current state's value changes is stored in a variable, Δ. Next, each of the current state's neighbors are slated for entry into the queue with priority Δ. If the queue is full, the entry with lowest priority is removed (Figure 3). P-LRTA* requires no initial world model; updates that spread to unseen states use the freespace assumption.

Once the current state's heuristic is updated, a series of prioritized updates begins. At most N states, where N is specified by the user, are taken from the top of the queue and updated using the procedure in Figure 2. If there are still states in the queue after the N updates, they remain in the queue to be used in the next planning phase.

Having completed the planning phase, the agent moves to the acting phase. The agent takes a single action, moving to the neighboring state s' with the minimum-valued $f(s, s') = c(s, s') + h(s')$, where $c(s, s')$ is the cost of traveling to s' and $h(s')$ is the current estimated cost of traveling from s' to the closest goal state (Figure 1).

5 Algorithm Properties

In this section we make general observations of the key features that make P-LRTA* different from other learning real-time search algorithms. Next, we discuss P-LRTA*'s convergence, as well as its space and time complexity.

Key Features and Design

Unlike Prioritized Sweeping's ϵ parameter, which restricts potentially small updates from entering the queue, P-LRTA* specifies a maximum queue size. This guarantees a strict limit on the memory and computational time used while enabling the algorithm to process arbitrarily small updates.

Because P-LRTA* specifies that the current state always be updated, P-LRTA* degenerates to Korf's LRTA*(d=1) [Korf, 1990] when the size of the queue is 0.

A P-LRTA* agent disallows duplicate entries in its queue, and retains the contents of its queue between acting phases. This design is a key difference between P-LRTA* and real-time search algorithms such as LRTA*(k) [Hernández and Meseguer, 2005b; 2005a] and Koenig's LRTA* [Koenig, 2004] since a P-LRTA* agent's local search space is not necessarily contiguous or dependent on the agent's current state. This property is beneficial, as a change to a single state's heuristic value can indeed affect the heuristic values of many, potentially remote, states. Empirically, limiting how far updates can propagate into unexplored regions of the state space does not significantly impact P-LRTA*'s performance.

The P-LRTA* algorithm we present specifies only a prioritized learning process and greedy action selection. But P-LRTA* can be modified to select actions in a way that incorporates techniques from other algorithms, such as increased lookahead within a specified radius [Bulitko and Lee, 2006], heuristic weighting [Shimbo and Ishida, 2003; Bulitko and Lee, 2006], or preferentially taking actions that take the agent through recently updated states [Koenig, 2004].

Theoretical Evaluation

Similar to Korf's LRTA* [Korf, 1990], P-LRTA* can be viewed as a special case of Barto *et al.*'s Trial-Based Real-Time Dynamic Programming (RTDP). In the following theorems, we outline why the theoretical guarantees of Trial-Based RTDP still hold for P-LRTA*, and explain P-LRTA*'s time and space complexity.

Theorem 1 P-LRTA* converges to an optimal solution from a start state $s_0 \in S$ to a goal state $s_g \in S$ when the initial heuristic is non-overestimating.

Our algorithm meets the criteria set out for convergence by Barto *et al.* in [Barto *et al.*, 1995]. It is shown that Trial-Based RTDP, and by extension P-LRTA*, converges to an optimal policy in undiscounted shortest path problems for any start state s_0 and any set of goal states S_g when there exists a policy that takes the agent to the goal with probability 1.

Theorem 2 P-LRTA*'s per-move time complexity is in $O(m \cdot log(q))$, where m is the maximum number of updates per time step and q is the size of the queue.

The primary source of P-LRTA*'s per-move time complexity comes from the m updates the algorithm performs, where m can be specified. Each of these m updates potentially results in b queue insertions, where b is the branching factor at the current state. Each of these insertions requires that we check whether that state is already queued ($O(1)$ using a hash table) and whether or not it has high enough priority to be queued ($O(log(q))$ using an ordered balanced tree).

Theorem 3 P-LRTA*'s memory requirements are of space complexity $O(|S|)$, where $|S|$ is the size of the state space.

P-LRTA* requires memory for both its environmental model and the states on the queue. Learning a model of an *a priori* unknown environment necessitates keeping track of a constant number of connections between up to $|S|$ states. Also, because duplicate states are disallowed, the maximum number of states in the queue, q, cannot exceed $|S|$.

6 Performance Metrics

Real-time search algorithms are often used when system response time and user experience are important, and so we measure the performance of search algorithms using the following metrics.

First-move lag is the time before the agent can make its first move. We measure an algorithm's first-move lag in terms of the number of states touched on the *first* move of the *last* trial; that is, after the agent has converged on a solution. This is in line with previous research in real-time search and enables us to compare new results to old results.

Suboptimality of the final solution is the percentage-wise amount that the length of the final path is longer than the length of the optimal path.

Planning time per unit of distance traveled indicates the amount of planning per step. In real-time multi-agent applications, this measure is upper bounded by the application constraints (*e.g.*, producing an action at each time step in a video game). An algorithm's planning time is measured in terms of the number of states touched per unit of distance traveled.

The *memory consumed* is the number of heuristic values stored in the heuristic table. This measure does not include the memory used to store the observed map, as the algorithms we run use the same map representation.

Learning heuristic search algorithms typically learn over multiple trials. We use *convergence execution cost* to measure the total distance physically traveled by an agent during the learning process. We measure this in terms of the distance traveled by the agent until its path converges, where distance is measured in terms of the cost c of traveling from one state to the next.

Note that to keep the measures platform independent, we report planning time as the number of states *touched* by an algorithm. A state is considered touched when its heuristic value is accessed by the algorithm. There is a linear correlation between the number of states touched and the wall time taken for our implementation.

7 Experimental Results

We ran search algorithms on path-planning problems on five different maps taken from a best-selling commercial computer game. The use of this particular testbed enabled comparison against existing, published data. The number of states in each map is 2,765, 7,637, 13,765, 14,098, and 16,142. We generated 2,000 problems for each map, resulting in a total of 10,000 unique path planning problems to be solved by each of the 12 competing parameterizations of learning real-time search algorithms (Table 1).

The agent incurs unit costs for moving between states in a cardinal direction, and costs of $\sqrt{2}$ for moving diagonally. All the algorithms experimented with use *octile distance* [Bulitko and Lee, 2006] as the initial heuristic h_0, which, for every state s, is the precise execution cost that would be incurred by the agent if there were no obstacles between s and the goal, s_g. The maps are initially unknown to each of the agents, and the uncertainty about the map is handled using the aforementioned freespace assumption. The visibility radius of each algorithm is set to 10, which limits each agent in what it knows about the world before it begins planning.

We compare P-LRTA* to five algorithms: LRA*, LRTA*(d=1), LRTS(d=10,γ=0.5,T=0), PR-LRTS with LRA* on the base level and LRTS(d=5,γ=0.5,T=0) on the first abstract level. The LRTS parameters were hand-tuned for low convergence execution cost. Since the size of Koenig's A*-defined strict search space is a similar parameter to P-LRTA*'s queue size, we compare against Koenig's LRTA* using local search spaces of size 10, 20, 30, and 40.

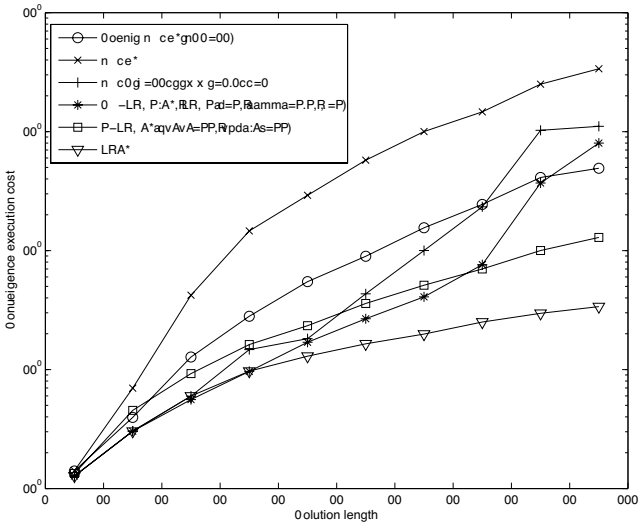

Figure 4: Convergence execution cost in semi-log averaged over 1,000 problems plotted against optimal solution length. LRTA*(d=1) has a large convergence execution cost, while P-LRTA*(queueSize=39,updates=40) has convergence execution cost comparable to LRA*.

Each algorithm's performance is tabulated for comparison in Table 1. LRA* and LRTA*(d=1) demonstrate extremes: LRA* has the smallest convergence execution cost and stores no heuristic values, but its first-move lag is the greatest of all algorithms because it plans all the way to the goal. LRTA*(d=1) has the largest convergence execution cost as a result of its simplistic update procedure, but its first-

Algorithm	Execution	Planning	Lag	Heuristic Memory	Suboptimality (%)
LRA*	**158.3** ± 1.3	49.8 ± 1.1	2255.2 ± 28.0	**0**	**0**
LRTA*(d=1)	9808.5 ± 172.1	**7.2** ± 0.0	**8.2** ± 0.0	307.8 ± 5.1	**0**
LRTS(d=10, γ=0.5, T=0)	3067.4 ± 504.4	208.6 ± 1.4	1852.9 ± 6.9	24.6 ± 1.2	0.9 ± 0.02
PR-LRTS(d=5,γ =0.5,T=0)	831.9 ± 79.1	57.6 ± 0.3	548.5 ± 1.7	9.3 ± 0.4	1.0 ± 0.02
Koenig's LRTA*(LSS=10)	2903.1 ± 51.5	70.9 ± 1.62	173.1 ± 0.3	424.5 ± 7.6	**0**
Koenig's LRTA*(LSS=20)	2088.6 ± 39.3	130.1 ± 3.5	322.1 ± 0.6	440.2 ± 8.2	**0**
Koenig's LRTA*(LSS=30)	1753.2 ± 32.4	188.6 ± 5.0	460.1 ± 1.1	448.8 ± 8.2	**0**
Koenig's LRTA*(LSS=40)	1584.4 ± 31.5	250.8 ± 7.5	593.9 ± 1.6	460.4 ± 8.7	**0**
P-LRTA*(queueSize=9, updates=10)	1236.0 ± 21.5	40.3 ± 0.9	**8.2** ± 0.0	339.0 ± 5.2	**0**
P-LRTA*(queueSize=19, updates=20)	708.2 ± 11.9	83.7 ± 1.8	8.3 ± 0.0	393.4 ± 5.8	**0**
P-LRTA*(queueSize=29, updates=30)	539.1 ± 8.8	129.7 ± 2.7	8.4 ± 0.0	444.7 ± 6.3	**0**
P-LRTA*(queueSize=39, updates=40)	462.4 ± 7.3	175.5 ± 3.6	8.3 ± 0.1	504.1 ± 7.2	**0**

Table 1: Results on 10,000 problems with visibility radius of 10. The results for LRA*, LRTA*, LRTS, and PR-LRTS are taken from [Bulitko *et al.*, 2005].

move lag and planning costs are extremely low. For each parameterization, P-LRTA* has a first-move lag comparable to LRTA*(d=1), and its convergence execution cost is even lower than algorithms that use sophisticated abstraction routines, such as PR-LRTS.

Figure 4 shows the average convergence execution cost of the algorithms on 1,000 problems of various lengths: P-LRTA*'s convergence execution cost is similar to LRA*'s. This indicates that heuristic learning in P-LRTA* is efficient enough that the expense of learning the heuristic function scales similarly with the expense of learning the *a priori* unknown map itself.

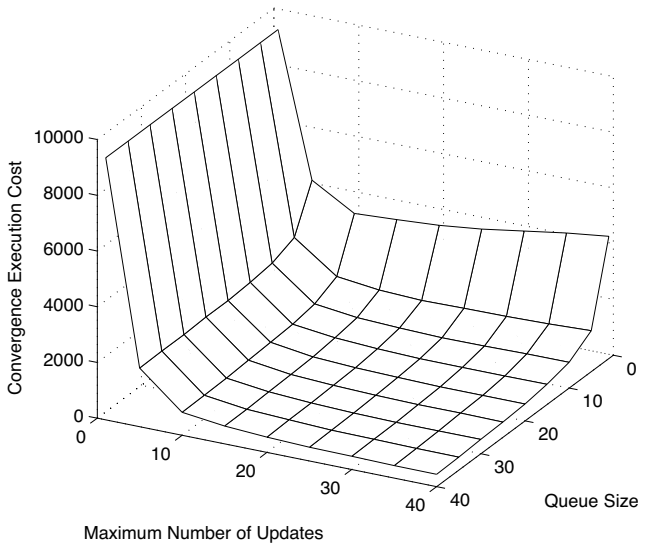

Figure 5: The impact of queue size and maximum number of updates on convergence execution cost.

Using the same set of 10,000 problems used to generate Table 1, we explore P-LRTA*'s parameter space. This experiment reveals that the queue size and the maximum number of updates per move exhibit some independence in how they affect performance. That is, when we increase the maximum number of updates with a fixed queue size, the convergence execution cost decreases; the same happens when we increase the queue size with a fixed maximum number of updates (Figure 5). This demonstrates that P-LRTA*'s parameters provide two effective ways of changing the algorithm to meet specific time and/or space requirements.

8 Future Work

Prioritized LRTA* is a simple, effective algorithm that invites further analysis. Further experimentation with randomized start-locations in the problems we explore could provide additional insight. We would also like to extend P-LRTA* to include dynamic parameterization, as in some settings it may be beneficial to use extra memory as it becomes available. P-LRTA* provides a convenient way to take advantage of free memory: the queue size can be dynamically adjusted.

P-LRTA* may benefit from a more sophisticated action-selection scheme than the purely greedy decision-making process we present. P-LRTA*'s simplicity makes it easy to experiment with a number of recent real-time search techniques that improve convergence time.

State abstraction has been successfully applied to real-time heuristic search [Bulitko *et al.*, 2005]. A hybrid approach that combines prioritized updates with state abstraction is likely to produce a search routine that is more powerful than either method alone. Another interesting direction is to extend P-LRTA* to handle dynamic environments by including focused exploration and allowing heuristic values to decrease.

9 Conclusions

Incremental search algorithms and real-time search algorithms meet different requirements. Incremental search algorithms such as LRA* converge quickly, but can suffer from arbitrarily long delays before responding. Real-time search works under a strict per-move computational limit, but its long on-line interactive learning process reduces its applicability when good solutions are needed from the start.

P-LRTA* has a response time on par with the fastest known real-time search algorithms. Meanwhile, its learning process converges in a time that scales with that of mere map discovery in LRA*. A P-LRTA* agent can learn twice as fast on the actual map as a state-of-the-art PR-LRTS agent learns on a smaller, abstract map.

As a step toward rapid-response algorithms that learn quickly, we presented a real-time heuristic algorithm that combines LRTA*'s aggressive initial heuristics and Prioritized Sweeping's ranking of state updates. P-LRTA* does not require the map *a priori* making it well suited to virtual reality trainers and computer games in which believable agents can only see limited portions of the world.

Acknowledgments

We would like to thank Nathan Sturtevant for developing HOG (Hierarchical Open Graph), the simulation framework we used to generate our empirical results. We would also like to thank Mitja Luštrek, David Thue, and several anonymous reviewers for their helpful comments. This work was supported by NSERC and iCORE.

References

[Barto *et al.*, 1995] Andrew Barto, Steven Bradtke, and Satinder Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72(1):81–138, 1995.

[Bulitko and Lee, 2006] Vadim Bulitko and Greg Lee. Learning in real time search: A unifying framework. *Journal of Artificial Intelligence Research*, 25:119 – 157, 2006.

[Bulitko *et al.*, 2005] Vadim Bulitko, Nathan Sturtevant, and Maryia Kazakevich. Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 1349 – 1354, Pittsburgh, Pennsylvania, 2005.

[Bulitko, 2004] Vadim Bulitko. Learning for adaptive real-time search. Technical report, Computer Science Research Repository (CoRR), 2004.

[Dini *et al.*, 2006] Don M. Dini, Michael Van Lent, Paul Carpenter, and Kumar Iyer. Building robust planning and execution systems for virtual worlds. In *Proceedings of the Artificial Intelligence and Interactive Digital Entertainment conference (AIIDE)*, Marina del Rey, California, 2006.

[Hart *et al.*, 1968] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Hernández and Meseguer, 2005a] Carlos Hernández and Pedro Meseguer. Improving convergence of LRTA*(k). In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains*, pages 69–75, Edinburgh, UK, 2005.

[Hernández and Meseguer, 2005b] Carlos Hernández and Pedro Meseguer. LRTA*(k). In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*, Edinburgh, UK, 2005.

[Koenig and Likhachev, 2002] Sven Koenig and Maxim Likhachev. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, pages 476–483, 2002.

[Koenig and Likhachev, 2006] Sven Koenig and Maxim Likhachev. Real-time adaptive A*. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 281–288, 2006.

[Koenig and Simmons, 1998] Sven Koenig and Reid Simmons. Solving robot navigation problems with initial pose uncertainty using real-time heuristic search. In *Proceedings of the International Conference on Artificial Intelligence Planning Systems*, pages 144 – 153, 1998.

[Koenig *et al.*, 2003] Sven Koenig, Craig Tovey, and Yuri Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*, 147:253–279, 2003.

[Koenig, 2001] Sven Koenig. Agent-centered search. *Artificial Intelligence Magazine*, 22(4):109–132, 2001.

[Koenig, 2004] Sven Koenig. A comparison of fast search methods for real-time situated agents. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pages 864 – 871, 2004.

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

[Moore and Atkeson, 1993] Andrew Moore and Chris Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13:103–130, 1993.

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

[Shue and Zamani, 1993a] Li-Yen Shue and Reza Zamani. An admissible heuristic search algorithm. In *Proceedings of the 7th International Symposium on Methodologies for Intelligent Systems (ISMIS-93)*, volume 689 of *LNAI*, pages 69–75. Springer Verlag, 1993.

[Shue and Zamani, 1993b] Li-Yen Shue and Reza Zamani. A heuristic search algorithm with learning capability. In *ACME Transactions*, pages 233–236, 1993.

[Shue *et al.*, 2001] Li-Yen Shue, S.-T. Li, and Reza Zamani. An intelligent heuristic algorithm for project scheduling problems. In *Proceedings of the Thirty Second Annual Meeting of the Decision Sciences Institute*, San Francisco, 2001.

[Sigmundarson and Björnsson, 2006] Sverrir Sigmundarson and Yngvi Björnsson. Value back-propagation vs backtracking in real-time heuristic search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Workshop on Learning For Search*, Boston, Massachusetts, 2006.

[Silver, 2005] David Silver. Cooperative pathfinding. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, Boston, Massachusetts, 2005.

[Stentz, 1995] Anthony Stentz. The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 1652–1659, 1995.