Prioritized-LRTA*: Speeding Up Learning via Prioritized Updates

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

Abstract

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

Introduction

Learning real-time search algorithms use heuristics – estimated distances from each state to the goal – to guide search, and learn by updating these heuristics to more accurate values. To be considered real-time in this context, these algorithms must be able to make progress immediately after the task has been specified. Instead of planning a route all the way to the goal, real-time search agents typically plan a partial route to the goal, execute the plan, and then re-plan. Interleaving planning and acting means that progress is being made in user-bounded time and that the controlled agent remains responsive.

As a motivating example, consider agents in real-time strategy (RTS) games. Path-planning in this setting is challenging because (1) there is a limited time allowed for decision making, (2) computation is distributed across many agents, and (3) the agent may have incomplete information about the world. Meeting these demands may require that the agents initially forfeit solution quality, but across trials, they learn better solutions.

We present a real-time search algorithm that makes learning more experience-efficient by prioritizing updates. 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, Bradtke, & Singh 1995). First, since states are updated one at a time, the new state values depend upon the order in which the updates occur; a judicious update ordering can make an individual update more effective. Second, a significant update to a state often affects the state's neighbors, and giving priority to these neighbors can focus computation on the most worthwhile updates.

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

Problem Formulation

A common task in real-time strategy games is directing an agent to move to a particular location. In this paper we focus on real-time path planning problems using a planar 8-connected grid world, a simplified version of environments seen in RTS games, for the purpose of illustration. However, the algorithm we propose is applicable to arbitrary search problems with explicit or implicit search spaces. Here we formalize the specific path-planning problem and motivate the metrics we use for evaluation.

The environment can be formally defined as the tuple

$$(S, A, c, s_0, S_g, h_0)$$

where S is a finite set of states, A is a set of deterministic actions, and $c(s, a)$ is the cost of performing action $a \in A$ in state $s \in S$. We use unit costs for traversing between states in the horizontal or vertical direction, and costs of $\sqrt{2}$ for diagonal movements. The agent begins in a particular state $s_0 \in S$ and its goal is to reach a state $s \in S_g$.

We use *octile distance* (Bulitko & Lee 2006) as the initial heuristic h_0, the precise execution cost that would be incurred by the agent if there were a clear path to the goal.

Some cells in the world are blocked; if an agent tries to move into a blocked cell the action fails and the agent remains in its current state. Blocked cells are discovered and remembered when they are within the agent's *visibility radius*, a natural number representing the maximum distance

at which the agent can sense the environment and update its transition model.

The agent deals with the uncertainty about whether a cell is blocked or not by believing that all unknown cells are unblocked; this belief is known as the *freespace assumption* (Koenig, Tovey, & Smirnov 2003). As the agent explores the map it can learn and remember which cells were blocked, and plan accordingly.

Real-time search algorithms are routinely used when system response time and user experience are important. We measure the performance of the algorithm using the metrics listed below.

- *First-move lag* measures the time before the agent can make its first move.
- *Suboptimality of the final solution* reflects the trade-offs made between the convergence speed and the quality of the solution.
- *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.
- The *memory consumed* is a measure of the number of heuristic values explicitly stored.
- Learning heuristic search algorithms typically learn over multiple trials. We use *convergence execution cost* to measure the total distance physically traversed by an agent during the learning process.

To be platform independent, we report measures of computational time in terms of the number of states touched. 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 time taken for our implementation of the real-time search algorithms we discuss.

Related Work

Real-time agents have the common property of being situated at any time in a single state, called the current state. To determine the best path from its current state to the goal, an agent has two approaches it can take: plan the entire way to the goal and then act, or make a partial plan, execute the plan, and then repeat until it reaches the goal. We review search algorithms for both of these methodologies.

Incremental A* plans a complete path from an agent's current state to the goal. The agent then tries to execute the plan. If the freespace assumption is incorrect and during execution there is an obstacle barring the path then the agent will stop and re-plan from its current state. Each plan made is the optimal path based on the agent's current knowledge. As each plan is generated with an A* search (Hart, Nilsson, & Raphael 1968), the time needed for the first move or any of the subsequent re-planning is of order $O(n\log n)$, where n is the number of states in a grid world. To decrease the time needed to re-plan, variations such as Dynamic A* (D*) (Stentz 1995) and D* Lite (Koenig & Likhachev 2002) use information from previous plans to quickly update their new plan. Although Dynamic A* and D* Lite speed up re-planning, they do not reduce the first move lag. For real-time strategy games it is important that this lag is small, since when multiple units are instructed to move simultaneously they are expected to comply promptly.

Korf's original Learning Real-Time A* (LRTA*) algorithm expanded single agent search by including the two-player game assumptions of limited search horizon and limited search time (Korf 1990). The result is an agent-situated search that alternates between planning and executing the plan. In its simplest version, LRTA* updates the heuristic value of the current state only with respect to its immediate neighbors. We shall refer to this version of LRTA* as LRTA*(d=1). If the goal is reachable from every state and the initial heuristic is admissible (*i.e.* non-overestimating) then LRTA* is guaranteed to converge to an optimal path (Korf 1990). The length of trials before convergence can oscillate unpredictably, causing an agent to behave in a seemingly irrational manner. Weighting of the heuristic values was added to reduce the path length instability (Shimbo & Ishida 2003; Bulitko & Lee 2006), but this also voids the guarantee of an optimal solution.

LRTA* is a special case of real-time dynamic programming (Barto, Bradtke, & Singh 1995), with the simplifying assumptions of a deterministic environment and a non-trivial initial value function (heuristic). A deterministic environment lets the agent make aggressive updates to a state's heuristic, such as the minimin update (Korf 1990) and the max-of-min update (Bulitko & Lee 2006).

Real-time learning search can also take advantage of an informative initial heuristic by developing sophisticated local search spaces and selectively updating states. We define a local search space (LSS) as the set of states touched by an agent in each planning stage. LRTA*(d=1) uses a myopic LSS, only looking at the current state's immediate neighbors (Korf 1990). This is not the only strategy available to an agent. A deeper lookahead gives an agent more information which it could potentially use to develop a better plan. Extended versions of LRTA* and LRTS (Bulitko & Lee 2006) can look at all of the states within a radius d of the current state. Rather than having a circular LSS, Koenig (2004) developed an algorithm that creates an LSS defined by a limited A* search from the current state toward the goal. This is only beneficial when the heuristic can be trusted, as deceptive values can quickly lead an agent astray.

Research has been directed toward reducing the convergence execution cost of real-time search by updating more than one state's heuristic values in each planning stage. One possibility is backtracking. SLA*, SLA*T (Shue & Zamani 1993a; 1993b; Shue, Li, & Zamani 2001), γ-Trap (Bulitko 2004), and LRTS (Bulitko & Lee 2006) all physically return to previously visited states, increasing the cost of a single trial but reducing the number of trials needed for convergence. LRTA*(k) (Hernández & Meseguer 2005b; 2005a) uses unprioritized mental backups to decrease the number of convergence trials, at the cost of using more memory. Koenig's algorithm (Koenig 2004) uses a Dijkstra-style update on all of the states expanded in its limited A* search, a computationally expensive step.

To reduce convergence execution cost PR-LRTS (Bulitko, Sturtevant, & Kazakevich 2005) builds a hierarchy of levels

PRIORITIZED-LRTA*(s)
while $s \neq s_g$ **do**
  $StateUpdate(s)$
  $n_{PS} = PS\_MAX\_UPDATE$
  while $queue \neq \emptyset$ and $n_{PS} > 0$ **do**
    $p = queue.pop()$
    if $p \neq s_g$ **then**
      $StateUpdate(p)$
      $n_{PS} \Leftarrow n_{PS} - 1$
    end if
  end while
  find neighbor s' with the lowest f(s, s') = c(s, s') + h(s)
  $s \Leftarrow s'$
end while

Figure 1: Prioritized-LRTA* has a planning phase in which it updates the current state, s, and then updates states from the queue until (1) the queue is empty, or (2) a maximum number of updates has been made. In the action phase a single greedy move is taken.

of abstraction and runs real-time search algorithms on each level. The search in the higher levels of abstraction is used to limit the lower-level searches to promising sections of the map, reducing the number of states explored in lower levels. The creation of the abstraction hierarchy causes the first move lag of this algorithm to be quite high.

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" (Barto, Bradtke, & Singh 1995). Prioritized Sweeping, an algorithm designed for reinforcement learning problems, performs asynchronous dynamic programming with a state's update ranked in order of priority (Moore & Atkeson 1993). A state has a high priority if it has a large potential change in its value function. To maintain the real-time guarantees of the algorithm a maximum of β updates can be made each planning phase, and only states with a potential update greater than ϵ are added to the queue. Prioritized Sweeping is shown to be more experience efficient than Q-learning and Dyna-PI (Moore & Atkeson 1993). Prioritized Sweeping's success motivates us to apply prioritized updates to real-time heuristic search.

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.

P-LRTA* can be thought of as having two phases: a planning phase and an action phase. In the planning phase the heuristic values of the current state and states on the queue are updated, and in the action phase the agent chooses which action to take and executes the action (Figure 1). Both

STATEUPDATE(s)
if s is not the goal **then**
  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**
    for all neighbors n of s **do**
      $AddToQueue(n, \Delta)$
    end for
    $h(s) \Leftarrow f(s, s')$
  end if
end if

Figure 2: StateUpdate(s) updates the value of state s to the lowest available $f(s, s') = c(s, s') + h(s')$, where $c(s, s')$ is the cost of traveling to s', and $h(s')$ is the current estimate of the distance from s' to the goal.

ADDTOQUEUE(s, Δ)
if $queue.notFull()$ and $queue.notIn(s)$ **then**
  $queue.add(s, \Delta)$
else
  $r \Leftarrow$ state with smallest Δ_r in $queue$
  if $\Delta_r < \Delta$ **then**
    $queue.remove(r)$
    $queue.add(s, \Delta)$
  end if
end if

Figure 3: AddToQueue(s, Δ) will insert state s into the finite-sized queue if the corresponding Δ is sufficiently large.

phases are described below.

In the planning phase, Prioritized-LRTA* updates the current state by considering only the state's immediate neighbors (Figure 2), like LRTA*(d=1). P-LRTA* places the updated state's neighbors on the queue, prioritized based on the size of the update, Δ; if the queue is full, the entries with lowest priority are removed (Figure 3). P-LRTA*, like LRTA*, requires no initial knowledge of the map, and updates that spread to unseen states use the freespace assumption. Moreover, when the queue size is zero, P-LRTA* is equivalent in execution to LRTA*(d=1).

Unlike Prioritized Sweeping's ϵ parameter, which restricts potentially small updates from entering the queue, Prioritized-LRTA* specifies a maximum queue size. In this way we guarantee a hard limit on the memory used and can add potentially small updates to the queue when it is not full.

After the current state's heuristic has been updated, the prioritized updates begin. At most PS_MAX_UPDATE states are taken from the top of the queue and updated using the procedure in Figure 1. If there are still states in the queue after all of the allowed updates have been made, they carry over to the next planning phase.

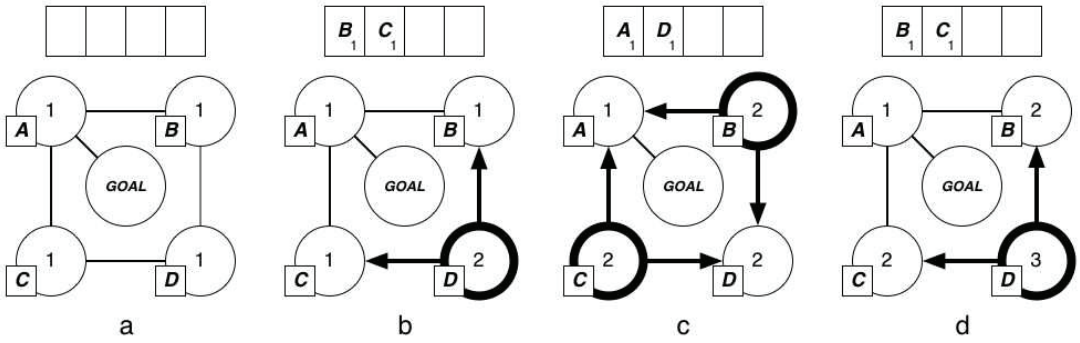

Figure 4: The Prioritized-LRTA* planning phase. An update (shown with a bold circle) occurs in a state adjacent to the goal and causes other states to be queued (as indicated with arrows) for updating. The states in the queue and their priorities are shown at the top of each diagram. The goal state is never queued.

The use of a priority queue for holding potential updates leads to a local search space that is not necessarily continuous. This is beneficial, as a change to a single state's heuristic value can affect the heuristic values of many, potentially remote, states.

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

Illustration of Prioritized-LRTA*'s Planning Phase

P-LRTA*'s execution can grow complicated in large state spaces, so we provide a simple five-state world to demonstrate prioritized updates. Figure 4a shows the initial configuration, with all heuristic values set to 1. The goal is an exception, as its heuristic value must always be 0. The transitions between states are bidirectional and the agent incurs a cost of 1 unit per move. We start with the agent situated in state D, the furthest state from the goal.

In Figure 4b, state D, the current state, receives an update as shown with a bold circle. Because D's heuristic value increases, its neighbors B and C are slated for entry into the queue, as shown with bold arrows. The queue now contains two elements with corresponding priorities of 1; the queue is shown in the top of the figure.

Figure 4c depicts updates based on queue entries. States B and C are removed from the queue and updated based on the values of neighboring states. Both states' heuristic values increase to 2 because the lowest-valued neighbor, state A, costs 1 to move to ($c = 1$) and its current heuristic is 1 ($h = 1$). Because B and C's heuristic values increase, their neighboring states A and D are enqueued.

The heuristic estimates for each state reach their optimal values in Figure 4d. States A and D are removed from the queue, but only state D receives a non-zero update: state A is 1 transition away from the zero-valued goal and maintains a heuristic value of 1. State D is 1 transition away from states B and C, which both have heuristic values of 2, and its heuristic value is updated to 3. This update results in neighboring states B and C being entered into the queue.

In this simple example, LRTA*(d=1) takes three trials before converging to optimal heuristic values, as opposed to P-LRTA*'s single trial.

Theoretical Analysis

Like Korf's LRTA* (Korf 1990), Prioritized-LRTA* can be considered a specific instance of Barto *et al.*'s Trial-Based RTDP. In this section, we show that the theoretical guarantees of Trial-Based RTDP still hold for P-LRTA*, and explain its time and space complexities.

Theorem 1 Prioritized-LRTA* converges to an optimal solution from a start state s to a goal state g when there exists a path from s to g and when the initial heuristic is admissible.

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

For a heuristic function to be *admissible* it must underestimate the cost from every state to the goal. This limits P-LRTA*'s applicability in dynamic environments. Like LRTA*, if any environmental change renders the heuristic function an overestimate, P-LRTA* is not guaranteed to converge. For example, introducing a short-cut through a wall may render the heuristic an overestimate since a P-LRTA* agent will believe that the path to the goal is longer than it actually is.

Theorem 2 Prioritized-LRTA*'s per-move time complexity is of $O(m \cdot q)$, 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 can potentially result in up to 8 insertions onto the queue. Each

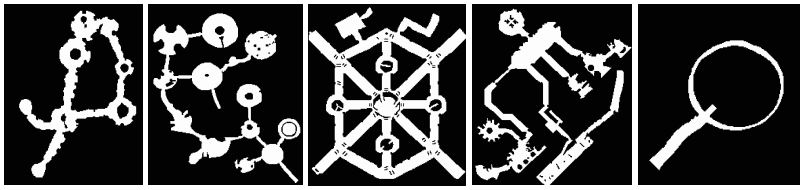

Figure 5: The five maps used to evaluate the algorithms.

of these insertions requires that we go through the list and check whether that state is already queued and whether or not it has high enough priority to be queued, which takes q comparisons. It is tempting to use a heap to maintain priority, but to limit the queue's size we use a sorted-order linked list; insertions are typically very fast due to the queue's small size.

Theorem 3 Prioritized-LRTA*'s memory requirements are of space complexity $O(s+q)$, where s is the size of the state space and q is the size of the queue.

The $O(s+q)$ memory requirements for P-LRTA* can be parameterized by manipulating q. Obviously a P-LRTA* agent must keep track of each of the q entries in the queue, but the algorithm must also learn a model of an *a priori* unknown environment, and keep track of the connections between each of s states.

Experimental Results

We compare Prioritized-LRTA* to several other state-of-the-art algorithms using path-planning problems from five different maps (Figure 5) taken from a best-selling commercial computer game, a winner of over 20 awards. The average map size is a 235×220 grid, and 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 parameterized algorithms (Table 1).

The maps are eight connected worlds, with the cost of physically moving between states set to one for horizontal or vertical movement and $\sqrt{2}$ for diagonal movement. The initial heuristic used is *octile distance* (Bulitko & Lee 2006). The maps are initially unknown to each of the agents, and the uncertainty about the map is dealt with by using the *freespace assumption* (Koenig, Tovey, & Smirnov 2003), in which unseen cells are assumed to be unblocked. The visibility radius of each algorithm is set to 10, which means an agent is able to determine if a cell is blocked or unblocked if the cell is less than or equal to 10 cells away.

We compare Prioritized-LRTA* to five algorithms: Incremental A*, LRTA*(d=1), LRTS(d=10,γ=0.5,T=0), PR-LRTS with incremental A* on the base level and LRTS(d=5,γ=0.5,T=0) on the first abstracted level. The LRTS parameters were hand tuned for low convergence execution cost. Since the size of Koenig's A* defined local 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.

We measure an algorithm's convergence execution cost in terms of the total distance traveled by the agent before its path converges, where distance is measured in terms of the cost of traveling from one state to the next, $c(s, s')$. An algorithm's planning cost is measured in terms of the number of states touched per unit of distance traveled.

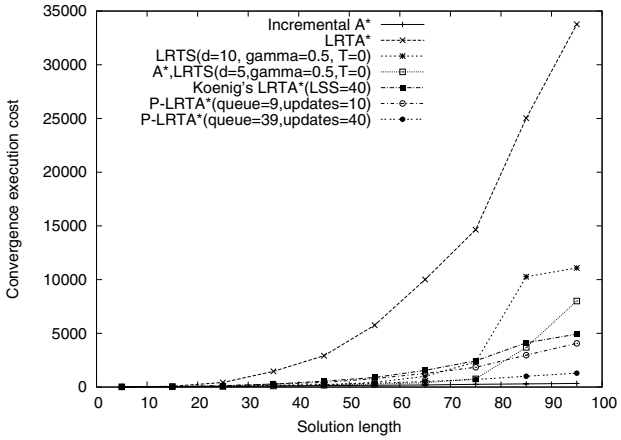

Figure 6: Average convergence of algorithms on 1,000 problems with different solution lengths. LRTA*($d = 1$) has a large convergence execution cost, while P-LRTA*($queue = 39, updates = 40$) has convergence execution cost comparable to Incremental A*.

To provide a platform-independent comparison of the algorithms we measure time in terms of how often the agent manipulates and stores a state's heuristic value. We call this the number of states touched, and it is a joint count of the number of states accessed during the planning phase and during and the action phase of the algorithm. Although the number of states touched is not strictly a measurement of time, our experiments have shown that this measurement is closely correlated to a search algorithm's reaction time. We measure an algorithm's first-move lag in terms of the number of states touched on the *first* move of the *last* trial. We measure the memory consumed in terms of the number of heuristic values stored for states whose heuristic value deviates from the original heuristic value, the octile distance from the state to the goal. And we measure the suboptimality of the final solution in terms of the percentage-wise amount the length of the final path is longer than the length of the optimal path.

Algorithm	Execution	Planning	Lag	Memory	Suboptimality (%)
Incremental A*	**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($d = 10, \gamma = 0.5, T = 0$)	3067.4 ± 504.4	208.6 ± 1.4	1852.9 ± 6.9	24.6 ± 1.2	0.9 ± 0.02
A*, LRTS($d = 5, \gamma = 0.5, T = 0$)	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*($queueSize = 9, updates = 10$)	1224.3 ± 21.3	59.7 ± 1.4	**8.2** ± 0.0	340.2 ± 5.2	**0**
P-LRTA*($queueSize = 19, updates = 20$)	606.7 ± 11.9	139.0 ± 3.2	8.3 ± 0.0	399.5 ± 5.8	**0**
P-LRTA*($queueSize = 29, updates = 30$)	530.1 ± 79.1	181.1 ± 4.0	8.3 ± 0.0	455.5 ± 6.4	**0**
P-LRTA*($queueSize = 39, updates = 40$)	458.3 ± 79.1	238.6 ± 5.2	8.3 ± 0.1	516.6 ± 7.2	**0**

Table 1: Results on 10,000 problems with visibility radius of 10. The results for Incremental A*, LRTA*, LRTS, and PR LRTS are taken from (Bulitko, Sturtevant, & Kazakevich 2005).

Each algorithm's performance is tabulated for comparison in Table 1. Incremental A* and LRTA*(d=1) demonstrate extremes: Incremental A* has the minimum 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-move lag and planning costs are extremely low. Ideally, we would like an algorithm that captures as much of these extremes as we can through efficient learning, and Prioritized LRTA* shows promise. For each parameterization, Prioritized-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 with A* and LRTS($d = 10, \gamma = 0.5, T = 0$).

Figure 6 shows the average convergence execution cost of the algorithms on 1,000 problems of various lengths: Prioritized-LRTA*'s convergence execution cost is comparable to Incremental A*'s, even as the solution length grows.

An interesting aspect of P-LRTA* is that the queue size and the maximum number of updates per move exhibit some independence in how they affect performance (Figure 7). When we increase the maximum number of updates with a fixed queue size, the convergence execution cost improves; the same happens when we increase the queue size with a fixed maximum number of updates. These two parameters provide two effective ways of changing the algorithm to meet specific time and space requirements.

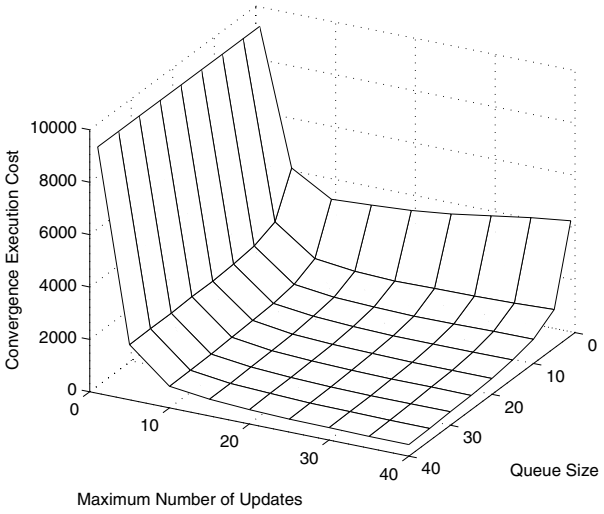

Figure 7: Impact of queue size and maximum number of updates on convergence execution cost.

Future Work

Prioritized LRTA* is a simple, effective algorithm that invites further analysis. In the future we will extend Prioritized-LRTA* to include dynamic parameterization, as in some settings it may be beneficial to use extra memory as it becomes available. Prioritized LRTA* provides a convenient way to take advantage of free memory: the queue size can be easily adjusted, even during execution.

State abstraction has been successfully applied to real-time heuristic search (Bulitko, Sturtevant, & Kazakevich 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.

As well, we are currently working on expanding P-LRTA* to handle dynamic environments by including focused exploration and allowing heuristic values to decrease.

Conclusions

In real-time strategy games, agents must respond quickly to a user's commands and exhibit reasonable behavior. As a step toward low-lag algorithms that learn quickly we present a real-time heuristic algorithm which combines LRTA*'s aggressive initial heuristics and Prioritized Sweeping's ranking of state updates. The algorithm does not require the map *a priori* and can be applied to future generations of real-time strategy games in which agents can only see limited portions of the world.

Empirical results show that Prioritized-LRTA* with a queue size of 40 has a convergence execution cost twenty times lower than LRTA*($d = 1$), three times lower than Koenig's algorithm, and two times lower than map-abstracting PR-LRTS. These significant convergence results do not come at the cost of increased lag, as Prioritized-LRTA*'s first move lag is the same as that of LRTA*($d = 1$).

Acknowledgments

We would like to thank Mitja Luštrek, David Thue, and several anonymous reviewers for their helpful comments.

References

Barto, A. G.; Bradtke, S. J.; and Singh, S. P. 1995. Learning to act using real-time dynamic programming. *Artificial Intelligence* 72(1):81–138.

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

Bulitko, V.; Sturtevant, N.; and Kazakevich, M. 2005. Speeding up learning in real-time search via automatic state abstraction. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1349 – 1354.

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

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 4(2):100–107.

Hernández, C., and Meseguer, P. 2005a. 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*.

Hernández, C., and Meseguer, P. 2005b. LRTA*(k). In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI)*.

Koenig, S., and Likhachev, M. 2002. D* Lite. In *Proceedings of the National Conference on Artificial Intelligence*, 476–483.

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

Koenig, S. 2004. 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 - Volume 2 (AAMAS'04)*, 864 – 871.

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

Moore, A., and Atkeson, C. 1993. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning* 13:103–130.

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

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

Shue, L.-Y., and Zamani, R. 1993b. A heuristic search algorithm with learning capability. In *ACME Transactions*, 233–236.

Shue, L.-Y.; Li, S.-T.; and Zamani, R. 2001. An intelligent heuristic algorithm for project scheduling problems. In *Proceedings of the Thirty Second Annual Meeting of the Decision Sciences Institute*.

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