Value Back-Propagation versus Backtracking in Real-Time Heuristic Search

Sverrir Sigmundarson and **Yngvi Björnsson**
Department of Computer Science
Reykjavik University
Ofanleiti 2
103 Reykjavik, ICELAND
{sverrirs01,yngvi}@ru.is

Abstract

One of the main drawbacks of the LRTA* real-time heuristic search algorithm is slow convergence. Backtracking as introduced by SLA* is one way of speeding up the convergence, although at the cost of sacrificing first-trial performance. The backtracking mechanism of SLA* consists of back-propagating updated heuristic values to previously visited states while the algorithm retracts its steps. In this paper we separate these hitherto intertwined aspects, and investigate the benefits of each independently. We present back-propagating search variants that do value back-propagation without retracting their steps. Our empirical evaluation shows that in some domains the value back-propagation is the key to improved efficiency while in others the retracting role is the main contributor. Furthermore, we evaluate learning performance of selected search variants during intermediate trial runs and quantify the importance of loop elimination for such a comparison. For example, our results indicate that the first-trial performance of LRTA* in pathfinding domains is much better than previously perceived in the literature.

Introduction

Learning Real-Time A* (Korf 1990), or LRTA* for short, is probably the most widely known real-time search algorithm. A nice property of the algorithm is that it guarantees convergence to an optimal solution over repeated trials on the same problem instance (given an admissible heuristic). In practice, however, convergence to an optimal solution may be slow, both in terms of the number of trials required and the total traveling cost. Over the years researchers have proposed various enhancements aimed at overcoming this drawback. These improvements include: doing *deeper lookahead* (Russell & Wefald 1991; Bulitko 2004; Koenig 2004), using *non-admissible heuristics*, although at the cost of forsaking optimality (Shimbo & Ishida 2003; Bulitko 2004), using more *elaborate successor-selection criteria* (Furcy & Koenig 2000), or incorporating a *backtracking* mechanism (Shue & Zamani 1993; 1999).

Backtracking affects the search in two ways. First, backtracking algorithms may choose to retract to states visited previously on the current trial instead of continuing further along the current path, resulting in an exploration strategy that differs substantially from LRTA*'s. Secondly, during this retracting phase changes in heuristic value estimates (h-values) get propagated further back. As these two aspects of backtracking have traditionally been closely intertwined in the real-time search literature, it is not fully clear how important role each plays. Only recently has the propagation of heuristic values been studied in real-time search as a stand-alone procedure (Koenig 2004; Hernández & Meseguer 2005b; 2005a; Rayner *et al.* 2006), building in part on earlier observations by Russell and Wefald (Russell & Wefald 1991) and the CRTA* and SLRTA* algorithms (Edelkamp & Eckerle 1997).

It can be argued that in some application domains value back-propagation cannot easily be separated from backtracking, because one must physically reside in a state to update its value. However, this argument does not apply for most agent-centered search tasks. For example, a typical application of real-time search is agent navigation in unknown environments. Besides the heuristic function used for guiding the search, the agent has initially only limited knowledge of its environment: knowing only the current state and its immediate successor states. However, as the agent proceeds with navigating the world it gradually learns more about its environment and builds an *internal model* of it. Because this model is kept in the agent's memory it is perfectly reasonable to assume that the agent may update the model as new information become available without having to physically travel to these states, for example to update the states' h-value. In that case, great care must be taken not to assume knowledge of successor of states not yet visited.

In this paper we take a closer look at the benefits of value back-propagation in real-time single-agent search. The main contributions of this paper are: 1) new insights into the relative effectiveness of back-propagation vs. backtracking in real-time search; in particular, in selected domains the effectiveness of backtracking algorithms is largely a side-effect of the heuristic value update, whereas in other domains their more elaborate successor-selection criterion is the primary contributor 2) an algorithmic formulation of back-propagating LRTA* search that assumes knowledge of only previously visited states; it outperforms LRTA* as well as its backtracking variants in selected application domains 3) quantification of the effects transpositions and loops have on real-time search solution quality in selected domains; for example, after eliminating loops LRTA* first-trial solution quality is far better than it has generally been perceived in the literature.

In the next section we briefly explain LRTA* and its most popular backtracking variants, using the opportunity to introduce the notation used throughout the paper. The subsequent section provides a formulation of value back-propagating LRTA* variants that use information of only previously seen states. The results of evaluating these and other back-propagating and backtracking real-time search algorithms are reported in our empirical evaluation section, but only after discussing some important issues pertaining to performance measurement. Finally we conclude and discuss future work.

LRTA* and Backtracking

Algorithm 1 LRTA*

```
 1: $s \leftarrow$ initial start state $s_0$
 2: $solutionpath \leftarrow \emptyset$
 3: while $s \notin S_g$ do
 4:     $h'(s) \leftarrow min_{s' \in succ(s)}(c(s,s') + h(s'))$
 5:     if $h'(s) > h(s)$ then
 6:         update $h(s) \leftarrow h'(s)$
 7:     end if
 8:     push $s$ onto top of $solutionpath$
 9:     $s \leftarrow argmin_{s' \in succ(s)}(c(s,s') + h(s'))$
10: end while
```

LRTA* is shown as Algorithm 1; s represents the state the algorithm is currently exploring, $succ(s)$ retrieves all the successor states of state s, and $c(s_1, s_2)$ is the transitional cost of moving from state s_1 to state s_2. At the beginning of each trial s is initialized to the start state s_0, and the algorithm then iterates until a goal state is reached (S_g is the set of all goal states). At each state s the algorithm finds the lowest estimated successor cost to the goal (line 4), but before moving to this lowest cost successor (line 9) the algorithm checks if a better estimate of the true goal distance was obtained, and if so updates the h-value of the current state (line 6). The variable $solutionpath$ keeps the solution found during the trial, which in LRTA* case is the same path as traveled (see discussion later in the paper). The LRTA* algorithm is run for multiple *trials* from the same start state, continually storing and using updated heuristic values from previous trials. The algorithm *converges* when no heuristic values (h-values) are updated during a trial. When used with an admissible heuristic it has, upon convergence, obtained an optimal solution. One of the main problems with LRTA* is that it can take many trials and much traveling for it to converge.

Search and Learning A* (SLA*), shown as Algorithm 2, introduced backtracking as an enhancement to LRTA* (Shue & Zamani 1993). It behaves identically to LRTA* when no h-value update is needed for the current state s. However, when a value update occurs SLA* does not move to the minimum successor state afterwards (as LRTA*), but rather backtracks immediately to the state it came from (lines 7 and 8). It continues the backtracking while the current state's heuristic value gets updated. Note that because of the backtracking SLA*'s $solutionpath$ does not store the path traveled but the best solution found during the trial.

Algorithm 2 SLA*

```
 1: $s \leftarrow$ initial start state $s_0$
 2: $solutionpath \leftarrow \emptyset$
 3: while $s \notin S_g$ do
 4:     $h'(s) \leftarrow min_{s' \in succ(s)}(c(s,s') + h(s'))$
 5:     if $h'(s) > h(s)$ then
 6:         update $h(s) \leftarrow h'(s)$
 7:         $s \leftarrow$ top state of $solutionpath$
 8:         pop the top most state off $solutionpath$
 9:     else
10:         push $s$ onto top of $solutionpath$
11:         $s \leftarrow argmin_{s' \in succ(s)}(c(s,s') + h(s'))$
12:     end if
13: end while
```

The main benefit of the SLA* algorithm is that its total travel cost to convergence it typically much less than LRTA*'s. Unfortunately, this comes at the price of very poor first-trial performance as all the traveling and learning takes place there. This is particulary problematic since a good first-trial performance is a crucial property of any real-time algorithm. SLA*T (Shue & Zamani 1999) somewhat alleviates this problem by introducing an user-definable learning threshold. The threshold, T, is a parameter which controls the cumulative amount of heuristic updates that must occur before backtracking is allowed. That is, SLA*T only backtracks after it has overflowed the T parameter. Although this does improve the first-trial behavior compared to SLA*, the first-trial performance is still poor when compared to LRTA*.

LRTA* and Value Back-Propagation

Algorithm 3 PBP-LRTA*

```
 1: $s \leftarrow$ initial start state $s_0$
 2: $solutionpath \leftarrow \emptyset$
 3: while $s \notin S_g$ do
 4:     $h'(s) \leftarrow min_{s' \in succ(s)}(c(s,s') + h(s'))$
 5:     if $h'(s) > h(s)$ then
 6:         update $h(s) \leftarrow h'(s)$
 7:         for all states $s_b$ in LIFO order from the
            $solutionpath$ do
 8:             $h'(s_b) \leftarrow min_{s' \in succ(s_b)}(c(s_b,s') + h(s'))$
 9:             if $h(s_b) >= h'(s_b)$ then
10:                 break for all loop
11:             end if
12:             $h(s_b) \leftarrow h'(s_b)$
13:         end for
14:     end if
15:     push $s$ onto top of $solutionpath$
16:     $s \leftarrow argmin_{s' \in succ(s)}(c(s,s') + h(s'))$
17: end while
```

SLA*'s backtracking mechanism serves two roles: firstly to back-propagate newly discovered information (in the form of updated h-values) as far back as possible and secondly to offer the search the chance to reevaluate its previous actions given the new information. A natural question to ask is how important part each of the two roles plays in reducing the overall traveling cost.

An algorithm that only performs the value back-propagation role of SLA*, not the backtracking itself, is illustrated as Algorithm 3. The algorithm, which we call *partial back-propagating LRTA** (PBP-LRTA*), is identical to LRTA* except with additional code for back-propagating changes in heuristic values (lines 5 to 14). This code is invoked when a heuristic value is updated in the current state s. It back-propagates the heuristic values as far up the solution path as needed, before continuing exploration from state s in the same fashion as LRTA* does.

PBP-LRTA* stops the back-propagation once it reaches a state on the $solutionpath$ where the heuristic value does not improve. This may, however, be misguided in view of transpositions in the state space. It could be beneficial to continue the back-propagation further up the solution path even though no update takes place in a given state; an update might occur further up the path (Figure 1 shows an example illustrating this). An algorithm for doing full back-propagation is identical to PBP-LRTA* except that lines 9 to 11 are removed. We call such a search variant *full back-propagating LRTA**, or FBP-LRTA* for short. Since both PBP-LRTA* and FBP-LRTA* use the original LRTA* policies for successor selection and heuristic value updating they retain the same properties as LRTA* regarding completeness and optimality.

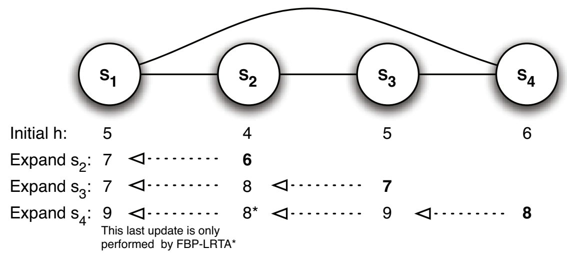


Figure 1: **Back-propagation stopping criteria example**
The search starts in state s_1 and travels through to state s_4, the initial heuristic for each state is given in the first line below the image. Edges imply connectivity between states, all edges have unit-cost. (1) When state s_2 is first expanded its h-value is updated from 4 to 6. This update is then back-propagated and s_1 updated to 7. (2) Next s_3 is expanded, its h-value gets updated from 5 to 7 and back-propagation is again triggered. An important thing happens now when the back-propagation updates s_1 since the estimated h-value of s_4 determines that the h-value of s_1 becomes 7. (3) When the search finally expands state s_4 its h-value is updated to 8, the back-propagation phase then updates s_3 to 9. However s_2 does not require an update since it uses s_1 estimate and keeps its value of 8. Here our PBP-LRTA* terminates its back-propagation (the point is marked with an asterisks). However since s_1 h-value was based on a now outdated h-estimate of state s_4 it still needs updating. When running FBP-LRTA*, state s_1 however gets updated to a more correct value of 9.

Table 1: First-Trial Statistics
The numbers represent the cost of the returned solution. A detailed description of the experimental setup is found in the experimental result section.

Baldur's Gate Maps			
	Average Solution Cost		
	With Loops	No Loops	% of Cost
FBP-LRTA*	508	89	17.5%
PBP-LRTA*	3,139	97	3.1%
LRTA*	3,610	90	2.5%
SLA*	71	71	100.0%
SLA*T(100)	110	80	72.7%
SLA*T(1000)	314	85	27.1%
8-puzzle			
	Average Solution Cost		
	With Loops	No Loops	% of Cost
FBP-LRTA*	388	111	28.6%
PBP-LRTA*	275	81	29.5%
LRTA*	380	61	16.1%
SLA*	20	20	100.0%
SLA*T(100)	96	41	42.7%
SLA*T(1000)	380	61	16.1%

Although formulated very differently, the FBP-LRTA* algorithm is closely related to the recently published LRTA*(∞) algorithm (Hernández & Meseguer 2005b). For the special case of undirected state spaces both algorithms would update heuristic values in the same set of states (the number of back-propagation steps in FBP-LRTA* could similarly be parameterized to be bounded by a constant).

Measuring Performance

When evaluating the performance of a real-time search algorithm it is important to measure both its computational efficiency and the quality of the solutions it produces.

Efficiency is typically measured as the agent's travel (or execution) cost. This cost does not include state expansions done in the planning phase, only the cost of the actions actually performed by the agent in the real world. Note that in all our discussion we make the assumption that this travel cost dominates the execution time, and the time overhead of the additional back-propagation will therefore be negligible. The number of trials to convergence is also typically used as a metric of efficiency, but the accumulated traveling cost over all trials is more representative in our view. Furthermore, it is an important criterion for a real-time algorithm to be able to produce *a* solution reasonably fast. This is measured by the first-trial travel cost metric.

Different real-time algorithms may produce solutions of different cost; the quality of the produced solutions can be measured by their relative cost compared to an optimal solution. Similarly as for the efficiency, we are interested in the quality of produced solutions both after the first and the final trial. All the algorithms we experiment with here use an admissible heuristic and are guaranteed to converge to an optimal solution. Consequently their final solution quality is the same. However, the intermediate solution quality may differ significantly from one algorithm to the next.

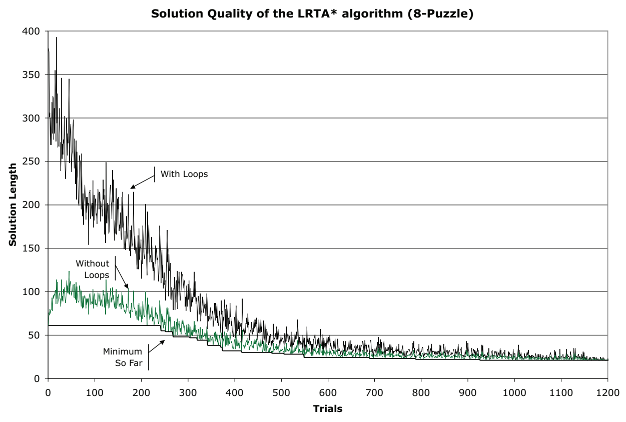


Figure 2: Effects of solution path loop elimination in the 8-puzzle.

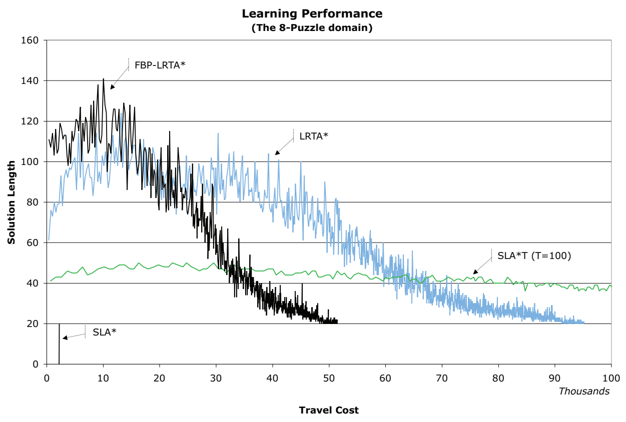


Figure 3: FBP-LRTA*, LRTA* and SLA*T(100) learning performance on the 8-puzzle domain.

The cost of the path traveled in a trial is not a good metric of the quality of the solution found in that trial. One reason for this is that real-time algorithms may wander in loops and repeatedly re-expand the same states (Korf 1990; Shimbo & Ishida 2003). Whereas the cost of traversing these loops rightfully count towards the travel cost, the loops themselves are clearly superfluous in the solution path and should be removed. Not all algorithms are affected equally by loops. For example, the SLA* algorithm is immune because of its backtracking scheme, whereas LRTA* is particularly prone. Therefore, when comparing solution quality of different algorithms it is important to eliminate loops from the solution path to ensure a fair comparison. This can be done either online or as a post-processing step after each trial.

Table 1 shows clearly how profound the effect of loop elimination can be. The extreme case in our experiments was LRTA* pathfinding game maps. After eliminating loops from the first-trial solutions, the remaining solution path length became only 2.5% of the path traveled. The resulting paths were then on average only sub-optimal by 27%. In this domain the true first-trial solution quality of LRTA* is clearly much better than it has generally been reported in the literature.

Similar effect, although not as profound, is also seen in other domains as shown in Figure 2. In the early trials there is a large difference in solution costs depending on whether loops are eliminated or not, although on later trials the two gradually converge to the same optimal path. For comparison, we include a line showing the best solution found so far. If the algorithm were to stop execution at any given moment, this would be the best solution path it can return.

A direct performance comparison of different algorithms can be problematic, even when run on the same problem set. For example, some algorithm may produce sub-optimal solutions, either because they are inadmissible or they do not converge in time. Because the final solution quality may then differ, one cannot look at the travel cost in isolation — there is a trade-off. The same applies if one wants to compare the relative efficiency of different algorithms during intermediate trials. To be able to do this we look at the solution cost as a function of the accumulated travel cost. This learning performance metric is useful for relative comparison of different algorithms and is a indicator of the algorithms' real-time nature. For example, in the domain shown in Figure 3 the SLA* T(100) algorithm quickly establishes a good solution but then converges slowly, whereas both LRTA* and FBP-LRTA* although starting off worse, in the end learn faster given the same amount of traveling. The figure also shows that FBP-LRTA* makes a better use of its learned values than LRTA* (the steep decent in its solution length is a clear indicator).

Experimental Results

To investigate the relative importance of backtracking vs. value back-propagation we empirically evaluated the PBP-LRTA* and FBP-LRTA* algorithms and contrasted them with LRTA*, SLA* and SLA*T in three different domains. The SLA*T algorithm was run with T values of 100 and 1000, respectively.

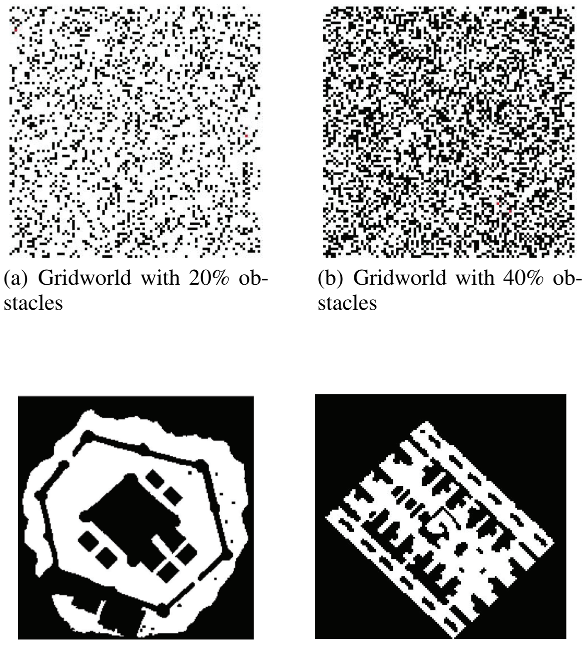
The first domain was path-finding in the Gridworld. The grids were of size 100x100 using three different obstacle ratios: 20%, 30%, 40%. One hundred randomly generated instances were created for each obstacle ratio. The second domain was also a path-finding task, but now on eight maps taken from a commercial computer game to provide a more realistic evaluation (Björnsson *et al.* 2003). Figure 4 shows some of the pathfinding maps that were used. For each of the eight maps 400 randomly chosen start-goal state pairs were used. The third domain used was the sliding tile puzzle (Korf 1985); 100 different puzzle instances were used. In all domains the Manhattan-distance heuristic was used (in the path-finding domains we only allowed 4-way tile-based movement). The above domains were chosen because they have all been used before by other researches for evaluating real-time algorithms and thus serve as a good benchmark for our research (Shimbo & Ishida 2003; Korf 1990; Bulitko & Lee 2005).

Table 2: Results from the pathfinding domains

Baldur's Gate Maps				
	Averaged Totals		First-trial	
	Travel Cost	Trials Conv.	Travel Cost	Sol. Len.
SLA*	17,374	1.81	17,308	71
FBP-LRTA*	19,695	63.40	508	89
SLA*T(100)	29,518	49.10	15,621	80
PBP-LRTA*	32,724	69.93	3,139	97
SLA*T(1000)	51,559	109.63	14,026	85
LRTA*	59,916	167.10	3,610	90
Gridworld with random obstacles				
	Averaged Totals		First-trial	
	Travel Cost	Trials Conv.	Travel Cost	Sol. Len.
FBP-LRTA*	8,325	35.32	389	102
SLA*	11,030	1.98	10,947	82
PBP-LRTA*	17,055	43.87	1,384	103
SLA*T(100)	17,705	46.67	9,223	91
SLA*T(1000)	24,495	69.90	8,404	97
LRTA*	29,760	90.67	2,237	102

Table 3: Results from the sliding-tile domains

8-puzzle				
	Averaged Totals		First-trial	
	Travel Cost	Trials Conv.	Travel Cost	Sol. Len.
SLA*	2,226	1.95	2,205	20
FBP-LRTA*	39,457	141.61	388	111
PBP-LRTA*	40,633	146.71	275	81
LRTA*	73,360	256.44	380	61
SLA*T(1000)	77,662	253.99	380	61
SLA*T(100)	149,646	202.77	651	41


(a) Gridworld with 20% obstacles

(b) Gridworld with 40% obstacles

(c) A large open-space Baldur's Gate map

(d) A corridor-room based Baldur's Gate map

Figure 4: A sample of the maps used for the pathfinding domains. Above is a sample of the Gridworld domain, below two of the eight Baldur's Gate maps used. The black areas represent obstacles.

Table 2 shows how the real-time search algorithms perform in the two path-finding domains. For each algorithm we report the total travel cost, the number of trials to convergence, the first-trial travel cost, and the solution length (with loops removed). Each number is the average over all test instances of the respective domain.

Both our value back-propagation algorithms outperform LRTA* significantly in the pathfinding domains, converging faster in terms of both number of trials and total travel cost. For example, FBP-LRTA* reduces the average number of trials to convergence on the Baldur's Gate maps by more than 100 (reduction of 62%). Its first-trial performance is also much better than LRTA*'s; an equally good solution is found on average using only a fraction of the search effort. Overall the FBP-LRTA* total traveling cost is roughly the same as SLA*'s, which is somewhat surprising because in the literature SLA* has been shown to consistently outperform LRTA*. Our result indicates however that the back-propagation of heuristic values, as opposed to backtracking, is largely responsible for the improved performance in pathfinding domains. Furthermore, FBP-LRTA* achieves this, unlike SLA*, while keeping its real-time characteristics by amortizing the learning over many trials. The new value back-propagation algorithms successfully combine the good properties of SLA* and LRTA*: SLA*'s short travel cost and fast convergence and LRTA*'s short first-trial delay and iterative solution approach.

Table 3 gives the same information as found in the previous table, but for the sliding-tile puzzle. In the 8-puzzle domain SLA* is clearly superior to the other algorithms when evaluated by total travel cost. This result is consistent with what has previously been reported in the literature (Bulitko & Lee 2005). In this domain backtracking, as opposed to only doing value back-propagation, is clearly advantageous. Also of interest is that the FBP-LRTA* and PBP-LRTA* algorithms perform almost equally well, contrary to the pathfinding domains where FBP-LRTA* is superior. This can be explained by the fact that there are relatively few transpositions in the sliding-tile-puzzle domain compared to the two pathfinding domains. Thus, the benefit of continuing the back-propagation is minimal. Also, somewhat surprisingly the first-trial cost of PBP-LRTA* is superior to FBP-LRTA*. We have currently no solid explanation for this. Preliminary results using the 15-puzzle also indicate that the overall benefits of value back-propagation are relatively small. SLA* was the only algorithm that was successful in that domain. It converged in 95 out of the 100 15-puzzle test cases using an upper-limit of 50 million states traveled, whereas the other algorithms all failed to converge even on a single problem.

Conclusions

In this paper we studied the effectiveness of backtracking versus value back-propagation in selected single-agent search domains. The good performance of backtracking algorithms like SLA* has often been contributed to the more elaborate successor-selection criteria. We showed that this is not true in general. For example, in pathfinding domains the performance improvement is mainly due to the effects of back-propagating updated heuristics, not the backtracking. The FBP-LRTA* search variant exhibited the best overall performance of all the real-time search algorithms we tried. Furthermore, in this domain the back-propagation variants successfully combine the nice properties of SLA* and LRTA*: SLA*'s low travel cost and LRTA*'s short first-trial delay and iterative solution approach.

On the other hand, contrary to the pathfinding domains, back-propagation is much less effective in the sliding tile puzzle. It showed some benefits on the 8-puzzle, but our preliminary results on the 15-puzzle indicate diminishing benefits. The different exploration criterion used by backtracking seems to be have far more impact than value updating. This poses an interesting research question of what properties of a problem domain favor backtracking versus value back-propagation. We suspect that complexity of the domain and the frequency of transpositions is in part responsible, but based on the evidence we have at this stage it is too premature to speculate much and we leave that for future research.

We also discussed the importance of eliminating loops from solutions paths prior to comparing different algorithms, and quantified the effects this had in our test domains. For example, LRTA* first-trial performance is much better in pathfinding domain than has generally been perceived in the literature. We also looked at the learning performance of selected real-time algorithms on intermediate trials.

There is still much more work that needs to be done to better understand the mutual and separate benefits of back-propagation and backtracking. Such investigation opens up many new interesting questions. There is clearly scope for new search variants that better utilize the benefits of both approaches by adapting to different problem domains.

Acknowledgments

This research has been supported by grants from The Icelandic Centre for Research (RANNÍS) and by a Marie Curie Fellowship of the European Community programme *Structuring the ERA* under contract number MIRG-CT-2005-017284. We also thank the anonymous reviewers for their comments.

References

Björnsson, Y.; Enzenberger, M.; Holte, R.; Schaeffer, J.; and Yap, P. 2003. Comparison of different abstractions for pathfinding on maps. *Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 03)* 1511–1512.

Bulitko, V., and Lee, G. 2005. Learning in real time search: A unifying framework. *Journal of Artificial Intelligence Research* 24.

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

Edelkamp, S., and Eckerle, J. 1997. New strategies in learning real time heuristic search. S. Edelkamp, J. Eckerle, New strategies in learning real time heuristic search, in: On-line Search: Papers from AAAI Workshop, Providence, RI, AAAI Press, 1997, pp. 30–35.

Furcy, D., and Koenig, S. 2000. Speeding up the convergence of real-time search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI/IAAI)*, 891–897.

Hernández, C., and Meseguer, P. 2005a. Improving convergence of LRTA*(k). In *In Proceedings of Workshop on Planning and Learning in A Priori Unknown or Dynamic Domains IJCAI-05*.

Hernández, C., and Meseguer, P. 2005b. LRTA*(k). In *In Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI-05*.

Koenig, S. 2004. A comparison of fast search methods for real-time situated agents. In *AAMAS '04: Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 864–871. Washington, DC, USA: IEEE Computer Society.

Korf, R. E. 1985. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence* 27(1):97–109.

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

Rayner, D. C.; Davison, K.; Bulitko, V.; and Lu, J. 2006. Prioritized-LRTA*: Speeding up learning via prioritized updates. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Workshop on Learning For Search*.

Russell, S., and Wefald, E. 1991. *Do the right thing: studies in limited rationality*. Cambridge, MA, USA: MIT Press.

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. 1993. An admissible heuristic search algorithm. In Komorowski, J., and Ras, Z. W., eds., *Methodologies for Intelligent Systems: Proc. of the 7th International Symposium ISMIS-93*. Berlin, Heidelberg: Springer. 69–75.

Shue, L.-Y., and Zamani, R. 1999. An intelligent search method for project scheduling problems. *Journal of Intelligent Manufacturing* 10:279–288.