Improving HLRTA*(k) [⋆]

Carlos Hernández[1] and Pedro Meseguer[2]

[1] UCSC, Caupolicán 491, Concepción, Chile
chernan@ucsc.cl
[2] IIIA, CSIC, Campus UAB, 08193 Bellaterra, Spain.
pedro@iiia.csic.es

Abstract. Real-time search methods allow an agent to move in unknown environments. We provide two enhancements to the real-time search algorithm HLRTA*(k). First, we give a better way to perform bounded propagation, generating the HLRTA*$_{LS}(k)$ algorithm. Second, we consider the option of doing more than one action per planning step, by analyzing the quality of the heuristic found during lookahead, producing the HLRTA*(k, d) algorithm. We provide experimental evidence of the benefits of both algorithms, with respect to other real-time algorithms on existing benchmarks.

1 Introduction

The classical heuristic search approach assumes that a solution can be computed off-line (i.e., by a systematic traversal of the search space), and once the whole solution is available, it is executed on-line. This approach is valid for some tasks (typically without uncertainty, with state spaces perfectly defined, in totally controlled environments). But it is not valid for other tasks when either (i) there is not enough information to compute a solution off-line (for instance, in unknown environments), or (ii) even if a complete solution could be computed, the task has some timing requirements and it cannot wait to compute the complete solution (for instance, a state space too large to be systematically explored). In some cases, both conditions hold (imagine a character in a video game, who has to react almost instantly to changes in a mostly unknown environment).

Real-time heuristic search is an alternative approach. Real-time search interleaves planning and action execution phases in an on-line manner, with an agent that performs the intended task. In the planning phase, one or several actions are planned, which are performed by the agent in the action execution phase. The planning phase has to be done in a limited, short amount of time. To satisfy this, real-time methods restrict search to a small part of the state space around the current state, which is called the *local space*. The size of the local space is small and independent of the size of the complete state space. Searching in the local space is feasible in the limited planning time. As result, the best trajectory inside the local space is found, and the corresponding action (or actions) are

[⋆] Supported by the Spanish REPLI-III project TIC-2006-15387-C03-01.

performed in the next action execution phase. The whole process iterates with new planning and action execution phases until a goal state is found.

This approach gives up the optimality of the computed solution. Obviously, if search is limited to a small portion of the state space, there is no guarantee to produce an optimal global trajectory. However, some methods guarantee that after repeated executions on the same problem instance (each execution is called a *trial*), the trajectory converges to an optimal path. To prevent cycling, real-time search methods update the heuristic values of the visited states.

The initial algorithms for real-time search were RTA* and LRTA* [9]. While RTA* performs reasonably well in the first trial, it does not converge to optimal paths. On the contrary, LRTA* converges to optimal paths with a worse performance in the first trial. Both approaches are combined in the HLRTA* algorithm [10]. Including the idea of bounded propagation, which propagates recursively a change in the heuristic of the current state up to a maximum of k states, the new HLRTA*(k) algorithm was proposed [6]. HLRTA*(k) keeps all good properties of HLRTA*, improving largely its performance in practice.

In this paper, we present further improvements to HLRTA*(k). First, we present an alternative method for bounded propagation. This new method implements propagation more efficiently than the initially proposed method. For instance, if a state should be updated, the initial method performs elementary updates, allowing the state to enter several times in the propagation queue. Now, all the updating is joined in a single operation. With this new method, we produce the new HLRTA*$_{LS}(k)$, which keeps all good properties of its predecessor and improves significantly its performance. Second, we consider the option of doing more than one action per planning step. There is some debate about the relative performance of planning one single action versus several actions per planning step, with the same lookahead. Our contribution is to consider the quality of the heuristic found during lookahead. If we find some evidence that the heuristic is not accurate, we plan one action only. Otherwise, we allow to plan several actions in this step. In addition, if some inaccuracy is detected during lookahead, it is repaired although it is not located at the current state.

The structure of the paper is as follows. First, we define precisely the problem, summarizing some of the most popular algorithms and explaining bounded propagation for the initial HLRTA*. We present our first contribution, the new HLRTA*$_{LS}(k)$ algorithm, which performs a single move per planning step. Then, we present and discuss our second contribution, the HLRTA*(k, d) algorithm that performs bounded propagation up to k states and is able to compute up to d moves per planning step. Both algorithms are experimentally evaluated on two benchmarks. Finally, we extract some conclusions from this work.

2 Background

Problem Definition. The state space is defined as (X, A, c, s, G), where (X, A) is a finite graph, $c : A \mapsto [0, \infty)$ is a cost function that associates each arc with a positive finite cost, $s \in X$ is the start state, and $G \subset X$ is a set of goal states.

X is a finite set of states, and $A \subset X \times X \setminus \{(x,x)\}$, where $x \in X$, is a finite set of arcs. Each arc (v,w) represents an action whose execution causes the agent to move from state v to state w. The state space is undirected: for any action $(x,y) \in A$ there exists its inverse $(y,x) \in A$ with the same cost $c(x,y) = c(y,x)$. The cost of the path between state n and m is $k(n,m)$. The successors of a state x are $Succ(x) = \{y|(x,y) \in A\}$. A heuristic function $h : X \mapsto [0,\infty)$ associates to each state x an approximation $h(x)$ of the cost of a path from x to a goal g where $h(g) = 0$ and $g \in G$. The exact cost $h^*(x)$ is the minimum cost to go from x to any goal. h is admissible iff $\forall x \in X, h(x) \leq h^*(x)$. h is consistent iff $0 \leq h(x) \leq c(x,w) + h(w)$ for all states $w \in Succ(x)$. A path $\{x_0, x_1, .., x_n\}$ with $h(x_i) = h^*(x_i), 0 \leq i \leq n$ is optimal.

RTA*/LRTA*. The pioneer and reference algorithms for real-time search are RTA* and LRTA* [9]. From the current state x, RTA* performs lookahead at depth d, and updates $h(x)$ to the max $\{h(x)$, 2nd min $[k(x,v) + h(v)]\}$, where v is a frontier state and $k(x,v)$ is the cost of the path from x to v. Then, the agent moves to y, successor of x, with minimum $c(x,y) + h(y)$. State y becomes the current state and the process iterates, until finding a goal. In finite state spaces with positive edge costs, finite heuristic values and where a goal state is reachable from every state, RTA* is correct, complete and terminates [9]. However, it does not converge to optimal paths when solving repeatedly the same instance, because of its updating strategy. Alternatively, the LRTA* algorithm behaves like RTA*, except that $h(x)$ is updated to the max $\{h(x)$, min $[k(x,v) + h(v)]\}$. This updating assures admissibility, provided the original heuristic was admissible, so the updated heuristic can be reused for the next trial. LRTA* is a correct and complete algorithm, that converges to optimal paths when solving repeatedly the same instance, keeping the heuristic estimates of the previous trial.

HLRTA*. RTA* works fine in the first trial but it does not converge to optimal paths. LRTA* converges but it performs worse than RTA* in the first trial. The HLRTA* algorithm [10] combines them as follows. It keeps for each visited state two heuristic values, h_1 and h_2, which correspond to the heuristic updating of LRTA* and RTA* respectively. In addition, it keeps in $d(x)$ the state where the agent moved from x (that is, $d(x)$ is the next current state from x). The interesting result here is that when search has passed through x, and it backtracks to x from $d(x)$ (that is, when it goes back to x through the same arc it used to leave) then h_2 estimate is admissible and it can be used instead of h_1 [10]. HLRTA* uses a heuristic $H(x) = max\{h_1(x), h_2(x)\}$ when h_2 is admissible, otherwise $H(x) = h_1(x)$. Since HLRTA* searches using admissible heuristics which are stored between trials, it converges to optimal paths in the same way that LRTA* does. Experimentally, HLRTA* does better than LRTA* in the first trial but it requires more trials than LRTA* to converge [2].

Bounded Propagation. Originally, real-time search algorithms updated the heuristic estimate of the current state only. In [5], the idea of *bounded propagation* was presented. Basically, it consists of propagating the change in the heuristic of the current state to its successor states. If some of them change their heuristic, these changes are propagated to its own successor states, and so on and so forth. Since the whole process could be long for a real-time context, a limit was proposed: after the first change, up to k states could be considered for further changes. Since propagation is limited up to k states, it is meaningful to consider which states are the most adequate to be updated. An option is to limit propagation to states already expanded. Other alternatives are discussed in [4].

This simple idea can be easily included in existing algorithms like LRTA* producing the LRTA*(k) version [5] (in fact, LRTA* is just a particular case of LRTA*(k) with $k = 1$). In practice, it has been shown to be very beneficial considering the effort to reach a solution in the first trial, and the number of trials to convergence. It also increases the solution stability before convergence. However, bounded propagation requires longer planning steps, since propagating to k states is computationally more expensive than propagating to one (the current) state. Nevertheless, benefits are important and the extra requirements on planning time are moderate, so if the application can accommodate longer planning steps, the use of bounded propagation is strongly recommended.

Considering HLRTA*, bounded propagation generates HLRTA*(k) [6] (again, HLRTA* is the particular case HLRTA*($k = 1$)) with similar benefits. Since HLRTA* keeps two heuristic estimates per visited state, it is worth noting that propagation is done on h_1, the heuristic that correspond to the updating of LRTA*. Performing propagation on h_2 may cause to lose the heuristic admissibility. This is due to the following fact. Let x, y and z be states. During propagation, $h_2(x)$ may go into $h_1(y)$, which after some steps is used to update $h_1(z)$. If propagation is long enough, $h_1(z)$ may go into $h_1(x)$, so the second minimum contribution appears in the first minimum without satisfying the conditions of admissibility for h_2 [10] (realize that the agent does not move during propagation). Without admissibility, convergence to optimal paths is not guaranteed.

3 HLRTA*$_{LS}(k)$

Bounded propagation was initially implemented using a propagation queue Q [6]. This implementation presented some weak points:

1. A state y may enter Q but, after reconsideration it may happen that $h(y)$ does not change. This is a wasted effort.
2. A state may enter Q more than once, making several updatings before reaching its final value. Would it not be possible to perform a single operation?
3. The order in which states enter Q, combined with the value of k parameter, may affect the final result.

These points are partially solved using a new implementation of bounded propagation, based on the notion of *local space* [7]. Formally, a local space is a pair (I, F), where $I \subset X$ is a set of interior states and $F \subset X$ is the set of frontier states, satisfying that F surrounds I immediate and completely, so $I \cap F = \emptyset$. The procedure to find the local space around the current state is as follows:

1. Set $I = \emptyset, Q = \{x\}$, where x is the current state.

```
procedure HLRTA*-LS(k)$(X, A, c, s, G, k)$
  for each $x \in X$ do $h_1(x) \leftarrow h_0(x)$; $h_2(x) \leftarrow 0$; $d(x) \leftarrow null$;
  repeat
    HLRTA-LS(k)-trial$(X, A, c, s, G, k)$;
    until $h_1$ does not change;
procedure HLRTA-LS(k)-trial$(X, A, c, s, G, k)$
  $x \leftarrow s$;
  while $x \notin G$ do
    if Changes?$(x)$ then
      $(I, F) \leftarrow$ SelectLS$(x, k)$;
      Dijkstra-shortest-paths$(I, F)$;
    HLRTA-LookaheadUpdate2min$(x)$;
    $y \leftarrow \text{argmin}_{w \in Succ(x)}[c(x, w) + \mathtt{H}(w, x)]$;
    execute$(a \in A$ such that $a = (x, y))$; $d(x) \leftarrow y$; $x \leftarrow y$;
function SelectLS$(x, k)$: pair of sets;
  $Q \leftarrow \langle x \rangle$; $F \leftarrow \emptyset$; $I \leftarrow \emptyset$; $cont \leftarrow 0$;
  while $Q \neq \emptyset \wedge cont < k$ do
    $v \leftarrow$ extract-first$(Q)$;
    $y \leftarrow \text{argmin}_{w \in Succ(v) \wedge w \notin I}[c(v, w) + \mathtt{H}(w, v)]$;
    if $h_1(v) < c(v, y) + \mathtt{H}(y, v)$ then
      $I \leftarrow I \cup \{v\}$; $cont \leftarrow cont + 1$;
      for each $w \in Succ(v)$ do
        if $w \notin I \wedge w \notin Q$ then $Q \leftarrow$ add-last$(Q, w)$;
    else if $I \neq \emptyset$ then $F \leftarrow F \cup \{v\}$;
  if $Q \neq \emptyset$ then $F \leftarrow F \cup Q$;
  return $(I, F)$;
procedure HLRTA-LookaheadUpdate2min$(x)$
  $z \leftarrow \text{arg 2nd min}_{v \in Succ(x)}[c(x, v) + \mathtt{H}(v, x)]$;
  if $h_2(x) < c(x, z) + \mathtt{H}(z, x)$ then $h_2(x) \leftarrow c(x, z) + \mathtt{H}(z, x)$;
function Changes?$(x)$: boolean;
  $y \leftarrow \text{argmin}_{v \in Succ(x)}[c(x, v) + \mathtt{H}(v, x)]$;
  if $h_1(x) < c(x, y) + \mathtt{H}(y, x)$ then return $true$; else return $false$;
function H$(v, from)$: real;
  if $d(v) = from$ then return $max\{h_1(v), h_2(v)\}$; else return $h_1(v)$;
```

Fig. 1. The HLRTA*$_{LS}(k)$ algorithm.

2. Loop until Q is empty or $|I| = k$ Extract a state w from Q. If w is a goal, exit loop. Otherwise, check by looking at $succ(w)$ that are not in I if $h(w)$ is going to change (if $h(w) < min_{v \in succ(w) - I} h(v) + c(w, v)$, we call this expression the updating condition). If so, include w in I, and $succ(w) - I$ in Q.
3. The set F surrounds I immediate and completely.

This procedure is called when x, the current state, satisfies the updating condition. Then, a local space (I, F) is computed around x. Observe that the number of interior states is upper bounded by k. Once the local space is determined, it is updated using a Dijkstra shortest paths procedure, updating the heuristic h_1

of interior states from the heuristic H of frontier states. If the initial heuristic is admissible, this updating process keeps admissibility [8].

When HLRTA* includes this form of bounded propagation, it is called HLRTA*$_{LS}(k)$. Its code appears in Figure 1. When I admits previously visited states only, this version is called HLRTA*$_{LS-path}(k)$. It not difficult to see that HLRTA*$_{LS}(k)$ inherits the good properties of HLRTA*(k), that is, it is correct, complete and terminates. Since admissibility is maintained, it also converges to optimal paths.

4 HLRTA*(k, d)

There is some debate about planning one action versus several actions per planning step, with the same lookahead. Typically, single-action planning produces trajectories of better quality (minor cost). However, the overall CPU time in single-action planning is usually longer than in the other approach, since the whole effort of lookahead produces a single move. Nevertheless, planning several actions is an attractive option that has been investigated [8], [1].

In unknown environments, the visibility range of an agent is the set of states around the current state that can be sensed by the agent. When planning several actions in unknown environments, moves are computed using the "free space assumption": if a state is not in the visibility range of the agent and there is no evidence that contains an obstacle, it is assumed to be feasible. When moves are performed, if an obstacle is found in one of these assumed feasible states, execution stops and a new planning phase starts.

Planning a single action per step is a conservative strategy. The agent has searched the local space and it has found the best trajectory in it. But from a global perspective, the agent is unsure whether this best trajectory effectively brings the agent closer to a goal state, or it follows a wrong path that will become apparent later. In this situation, the least commitment strategy is to plan a single action: the best move from the current state.

Planning several actions per step is a more risky strategy. Following the best trajectory in the local space is risky because (i) it might not be good at global level, and (ii) if it is finally wrong, since it includes several actions, it will require some effort to come back. Otherwise, if the trajectory is good, performing several moves in one step will bring the agent closer to the goal than a single move.

These strategies are two extremes of a continuum of possible planning strategies. We propose an intermediate option, that consist on taking into account the quality of the heuristic found during lookahead. If there is some evidence that the heuristic quality is not good at local level, we do not trust the heuristic values and plan one action only. Otherwise, if the heuristic quality is good in the local space, we trust it and plan for several actions. Specifically, we propose not to trust the heuristic when one of the following conditions holds:

1. the final state for the agent (= first state in OPEN when lookahead is done using A*) satisfies the updating condition,
2. there is a state in the local space that satisfies the updating condition.

```
procedure HLRTA*(k,d)($X, A, c, s, G, k, d$)
  for each $x \in X$ do $h_1(x) \leftarrow h_0(x)$; $h_2(x) \leftarrow 0$; $d(x) \leftarrow null$;
  repeat
    HLRTA(k,d)-trial($X, A, c, s, G, k, d$);
    until $h_1$ does not change;
procedure HLRTA*(k,d)-trial($X, A, c, s, G, k, d$)
  $x \leftarrow s$;
  while $x \notin G$ do
    $path \leftarrow$ A*($x, d, G$); $z \leftarrow$ last($path$);
    if Changes?($z$) then
      $(I, F) \leftarrow$ SelectLS($z, k$);
      Dijkstra-shortest-paths($I, F$);
      HLRTA-LookaheadUpdate2min($x$);
      $y \leftarrow \text{argmin}_{w \in Succ(x)}[c(x, w) + \texttt{H}(w, x)]$;
      execute($a \in A$ such that $a = (x, y)$); $d(x) \leftarrow y$; $x \leftarrow y$;
    else
      $x \leftarrow$ extract-first($path$);
      while $path \neq \emptyset$ do
        HLRTA-LookaheadUpdate2min($x$);
        $y \leftarrow$ extract-first($path$);
        execute($a \in A$ such that $a = (x, y)$); $d(x) \leftarrow y$; $x \leftarrow y$;
```

Fig. 2. The HLRTA*(k, d) algorithm. Missing procedures/functions appear in Fig. 1.

In both cases, we repair the inaccuracy of the heuristic, that is, we generate a local space around that state, we update the heuristic and this change is propagated by bounded propagation. This is an important point in our approach: as soon as one heuristic inaccuracy is detected, it is repaired and propagated.

These ideas are implemented in the HLRTA*(k, d) algorithm. It is based on HLRTA*, and it propagates heuristic updates up to k states [6]. In addition, it is able to plan either 1 or up to d actions per planning step. It includes:

– Lookahead using A*. Following [8], the lookahead required to plan more than one action per step is done using the well-known A* algorithm [3].
– Local space selection. When $h(x)$ of a state x in the lookahead changes, the local space around x is computed by the `SelectLS` procedure (Section 3).
– Propagation in local space. Once the local space is selected, propagation of heuristic changes into the local space is done using the Dijkstra shortest paths algorithm, as done by [8].

HLRTA*(k, d) is more that a novel combination of existing techniques. As new element, the algorithm determines the number of actions to plan depending on the quality of the heuristic found in the lookahead. If the heuristic value of some state found during lookahead satisfies the updating condition, lookahead stops, this change is propagated up to k states and one action is planned only. If no heuristic value satisfies the updating condition in the lookahead states, a sequence of d actions are planned. These actions are executed in the execution phase, taking into account that if an obstacle is found, the execution stops and a new planning phase starts.

The code of HLRTA*(k, d) appears in Figure 2. The central procedure is `HLRTA*(k,d)-trial`, that is executed once per trial until finding a solution. This procedure works at follows. First, it performs lookahead from the current state x using the A* algorithm. A* performs lookahead until (i) it finds an state which heuristic value satisfies the updating condition, (ii) if finds a state w such that $g(w) = d$, or (iii) it finds a solution state. In any case, it returns the sequence of states, *path*, that starting with the current state x connects with (i) the state which heuristic value satisfies the updating condition, (ii) a state w such that $g(w) = d$, or (iii) a solution state. Observe that *path* has at least one state x, and the only state that might change its heuristic value is `last`(*path*). If this state satisfies the updating condition, then this change is propagated: the local space is determined and updated using the shortest paths algorithm. Then, one action is planned, executed and the loop iterates. If `last`(*path*) does not change its heuristic value, then up to d actions are planned and executed.

Since the heuristic always increases, HLRTA*(k, d) completeness is guaranteed. If the heuristic is initially admissible, updating the local space with shortest paths algorithm keeps admissibility, so convergence to optimal paths is guaranteed. So HLRTA*(k, d) inherits the good properties of HLRTA*.

One might expect that HLRTA*(k, d) collapses into HLRTA*$_{LS}(k)$ when $d = 1$. However, this is not the case. When $d = 1$, these two algorithms basically differ in the following. If the heuristic of the current state satisfies the updating condition, HLRTA*$_{LS}(k)$ updates it and propagates this change in a local space constructed around the current state. In this case, HLRTA*$(k, 1)$ behaves exactly like HLRTA*$_{LS}(k)$. But if the heuristic of the current state does not change, HLRTA*$(k, 1)$ generates a local space using the A* algorithm, and if the heuristic of some state of this local space satisfies the updating condition, it is updated and this change is propagated in a local space around that state.

5 Experimental Results

We compare the performance of HLRTA*$_{LS}(k)$ and HLRTA*(k, d) with HLRTA*(k) [6] and LRTA* (version of Koenig) [8]. Parameter k is the size of the local space, where bounded propagation is performed; it is usually taken as the lookahead parameter for LRTA*. We have used the values $k = 5, 10, 20, 40, 80$. Parameter d is the upper limit on the number of planned actions per step for HLRTA*(k, d). We have used the values $d = 1, 2, 4, 6$. Benchmarks are four-connected grids where an agent can move one cell north, south, east or west, on which we use Manhattan distance as the initial heuristic. We use the following benchmarks:

1. Grid35. Grids of size 301×301 with a 35% of obstacles placed randomly. Here, Manhattan distance tends to provide a reasonably good advice.
2. Maze. Acyclic mazes of size 181×181 whose corridor structure was generated with depth-first search. Here, Manhattan distance could be very misleading.

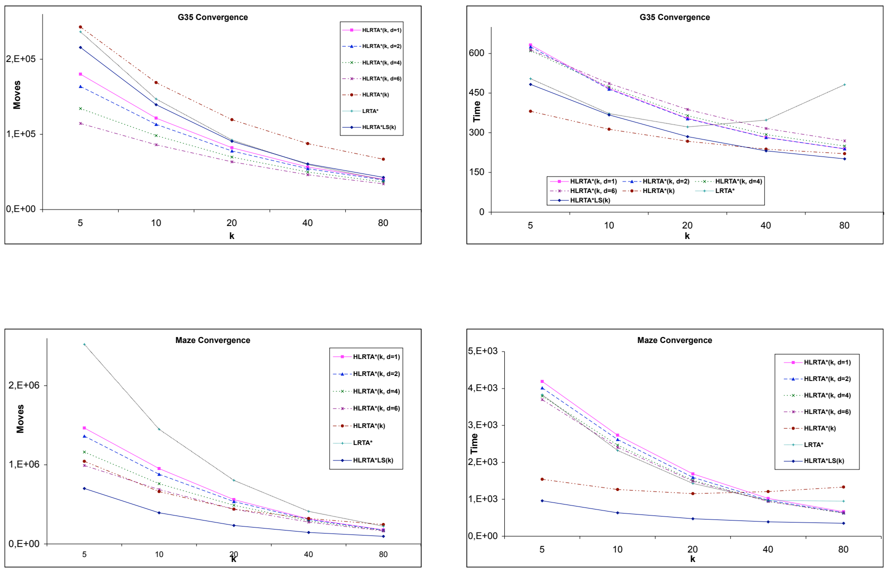

Fig. 3. Experimental results on Grid35 and Maze benchmarks: solution cost (left) and total planning time (right) for convergence of Grid35 (1st row) and Maze (2nd row).

In both benchmarks, the start and goal states are chosen randomly assuring that there is a path from the start to the goal. All actions have cost 1. The agent visibility radius is 1. We have obtained results for first trial and convergence to optimal trajectories. For space reasons, only convergence results are shown in Figure 3: solution cost (number of actions to reach the goal) and total planning time (in milliseconds), plotted against k, averaged over 1500 different instances.

Results for convergence on Grid35 indicate that solution cost decreases monotonically as k increases, and for HLRTA*(k,d) it also decreases monotonically as d increases. HLRTA*(k,d) versions obtain the best results for low lookahead, and all algorithms have a similar cost for high lookahead (except HLRTA*(k) which has a higher cost). Considering total planning time, all algorithms decrease monotonically with k except LRTA*, which first decreases and from $k = 20$ increases again. HLRTA*(k,d) versions require more time than HLRTA*(k) and HLRTA$*_{LS}(k)$, which are the fastest algorithms in the whole k range.

Results for convergence on Maze exhibit a slightly different behavior. Regarding solution cost, all algorithms decrease as k increases. HLRTA$*_{LS}(k)$ obtains the best cost for the whole k range. Regarding total planning time, for HLRTA*(k,d) versions it decreases monotonically as k increases, with little difference for d parameter. The interesting point here is that HLRTA$*_{LS}(k)$ is again the fastest algorithm in the whole k range.

From these results, we conclude that the main parameter is k, the lookahead size. For high lookahead ($k = 40, 80$), HLRTA*(k, d) with a low number of moves ($d = 1, 2$) or HLRTA*$_{LS}(k)$ offer the best trade-off between solution cost and planning time. For low lookahead ($k = 5, 10$), HLRTA*(k, d) versions offer the best solution cost, while HLRTA*$_{LS}(k)$ has better time requirements.

The Maze benchmark deserves special analysis. For this problem, the best algorithm is HLRTA*$_{LS}(k)$, unbeaten by the more sophisticated HLRTA*(k, d). We believe that this is due to the special structure of the benchmark, with many corridors that finalize in dead-ends. Apparently, a relatively simple strategy using the second min updating is enough to obtain very good results. More research is required to confirm this hypothesis.

6 Conclusions

We have presented two contributions to improve HLRTA*(k). First, a new method to implement bounded propagation, producing the new HLRTA*$_{LS}(k)$ algorithm. Second, a new approach to plan more than one action per step, analyzing the quality of the heuristic found during lookahead. This approach generates the new HLRTA*(k, d) algorithm. Both algorithms are correct, complete, terminate and converge to optimal trajectories after repeated executions on the same problem instance. Experimentally, we have observed that they achieve a good performance, improving over LRTA (version of Koenig) and HLRTA*(k). Apparently, the ability to plan a few moves per step is beneficial, provided these moves are of good quality. This is done by assessing the quality of the heuristic. We believe that the results on the Maze are due to its special structure.

References

1. V. Bulitko and G. Lee. Learning in real time search: a unifying framework. *Journal of Artificial Intelligence Research*, 25:119–157, 2006.
2. D. Furcy and S. Koenig. Combining two fast-learning real-time search algorithms yields even faster learning. In *Proc. 6th European Conference on Planning*, 2001.
3. P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Sys. Science & Cybernetics*, 2:100–107, 1968.
4. C. Hernandez and P. Meseguer. Improving convergence of lrta(k). In *IJCAI'05 Work. on Planning and Learning in a Priori Unknown or Dynamic Domais*, 2005.
5. C. Hernandez and P. Meseguer. Lrta*(k). In *Proc. IJCAI-05*, pages 1238–1243, 2005.
6. C. Hernandez and P. Meseguer. Propagating updates in real-time search: Hlrta*(k). In *Proc. CAEPIA'05, LNAI 4177, Ed. R. Marin*, pages 379–388, 2006.
7. C. Hernandez and P. Meseguer. Improving lrta*(k). In *Proc. IJCAI-07*, pages 2312–2317, 2007.
8. S. Koenig. A comparison of fast search methods for real-time situated agents. In *Proc. AAMAS-04*, pages 864–871, 2004.
9. R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42:189–211, 1990.
10. P. E. Thorpe. A hybrid learning real-time search algorithm. Master's thesis, Computer Science Dep., UCLA, 1994.