Propagating Updates in Real-Time Search: HLRTA*(k)[⋆]

Carlos Hernández and Pedro Meseguer

Institut d'Investigació en Intel.ligència Artificial
Consejo Superior de Investigaciones Científicas
Campus UAB, 08193 Bellaterra, Spain
{chernan, pedro}@iiia.csic.es

Abstract. We enhance real-time search algorithms with bounded propagation of heuristic changes. When the heuristic of the current state is updated, this change is propagated consistently up to k states. Applying this idea to HLRTA*, we have developed the new HLRTA*(k) algorithm, which shows a clear performance improvement over HLRTA*. Experimentally, HLRTA*(k) converges in less trials than LRTA*(k), while the contrary was true for these algorithms without propagation. We provide empirical results showing the benefits of our approach.

1 Introduction

Real-time search interleaves planning and action execution in an on-line manner. This allows to face search problems in domains with limited information, where it is impossible to perform the classical search approach: planning first the whole solution off-line, and then executing such solution. For these domains, interleaving planning and action execution is needed: planning explores some local search space selecting the best action from that limited exploration, and action execution carries out that action, causing changes in the world. The process repeats until achieving a solution. To assure completeness, real-time search has to record in a hash table those actions that have been executed. Often, there is an upper bound on the computing time that the planning phase can take.

Real-time search is also useful for domains with complete information whose search spaces are too large to be explored in practice. In such cases, real-time search is a suitable alternative to the impractical off-line search [7].

A number of algorithms have been proposed to perform real-time search. They are based on the following strategy. Assuming an initial finite heuristic values, the algorithm explores a search space local to the current state and updates its heuristic accordingly, backing up costs. Then, the algorithm moves to the next state where exploration/updating is repeated. The process is iterated until finding a goal state. Some of these algorithms can find optimal solutions: solving repeatedly the same problem instance performance improves and they converge eventually to optimal paths.

⋆ Supported by the Spanish REPLI project TIC-2002-04470-C03-03.

In the author's knowledge the first proposed real-time search algorithms were RTA* and LRTA* in the seminal work by Korf [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 [11].

Instead of updating the heuristic of one (the current) state only, we have proposed to propagate the heuristic change consistently up to k states [6]. We call this idea bounded propagation, and we have applied it to LRTA* [6,5], producing the LRTA*(k) algorithm. Experimental results show that LRTA*(k) causes large benefits on the first solution, convergence and solution stability with respect to LRTA*, at the extra cost of longer planning steps. In this paper, we apply bounded propagation to HLRTA*, producing the new HLRTA*(k) algorithm. This is not a direct extension of the work done in LRTA*(k): HLRTA* deals with two heuristics h_1 (always admissible) and h_2 (admissible under some circumstances) that require a careful handling, since propagation is done on admissible heuristic values. Experimental results show that bounded propagation causes the same kind of benefits on the first solution, convergence and solution stability. These results show that, for $k > 1$, HLRTA*(k) converges in less trials than LRTA*(k), while the contrary was true for the original algorithms.

The paper structure is as follows. In Section 2 we revise existing work on real-time search, with special emphasis on HLRTA*. In Section 3 we present the idea of bounded propagation. In Section 4, we describe the new HLRTA*(k) algorithm, showing its correctness and completeness. In Section 5, we provide experimental results of HLRTA*(k) on classical real-time benchmarks. Finally, in Section 6 we extract some conclusions from this work.

2 Real-Time Search: RTA*, LRTA* and HLRTA*

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 finite cost, s is the start state, and $G \subset X$ is the set of goal states. X is a finite set of states, and $A \subset X \times X - \{(x, x) | x \in X\}$ is a finite set of arcs. Each arc (v, w) represents an action whose execution causes the agent to move from v to 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 successors of a state x are $Succ(x) = \{y | (x, y) \in A\}$. A path $(x_0, x_1, x_2, \ldots)$ is a sequence of states such that every pair $(x_i, x_{i+1}) \in A$. The cost of a path is the sum of costs of the actions in that path. A heuristic function $h : X \mapsto [0, \infty)$ associates with each state x an approximation $h(x)$ of the cost of a path from x to a goal. $h^*(x)$ is the minimum cost to go from x to a goal. h is said to be admissible iff $h(x) \leq h^*(x)$, $\forall x \in X$. A path $(x_0, x_1, \ldots, x_n)$ with $h(x_i) = h^*(x_i)$, $0 \leq i \leq n$ is *optimal.*

RTA* works as follows. From the current state x, it 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, it moves to y, successor of x, with minimum $c(x, y) + h(y)$. This state becomes the current state and the process iterates, until eventually finding a goal. This process is called a trial.

RTA* is a correct and complete algorithm in finite state spaces with positive edge costs, finite heuristic values and where a goal state is reachable from every state [9]. However, it is unable to improve its performance when solving repeatedly the same instance. This is due to the 2nd min updating strategy. To solve this issue, the LRTA* algorithm was proposed [9]. It behaves like RTA*, except that $h(x)$ is updated to the max $\{h(x),\ \min\ [k(x,v)+h(v)]\}$. This updating rule assures admissibility, provided the original heuristic was admissible. Then, 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.

RTA* works fine in the first trial but there is no guarantee to convergence after successive trials to optimal paths. LRTA* converges but it performs worse than RTA* in the first trial. Would it not be possible to combine them? The answer is the HLRTA* algorithm [11]. HLRTA* keeps for each visited node 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 next current node 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 [11]. Since HLRTA* always keeps admissible heuristic estimates, they are stored between trials and it converges to optimal paths, in the same way that LRTA* does. Experimentally, HLRTA* requires more trials than LRTA* to converge [4].

The HLRTA* algorithm with lookahead at depth 1 and with h admissible appears in Figure 1. Like in [9], we assume the existence of $Succ$ and h_0 functions, which when applied to a state x generate its set of successors and its initial heuristic estimate, respectively. Procedure `HLRTA*` initializes the heuristic estimates h_1 and h_2 of every state to h_0 and 0 respectively. It also initializes $d(x)$ with $null$. Then it repeats the execution of `HLRTA-trial` until convergence of the heuristic function, i.e., until h_1 does not change anymore. At this point, an optimal path has been found. Procedure `HLRTA-trial` performs a solving trial on the problem instance. It initializes the current state x with the start s, and executes the following loop until finding a goal. First, it performs lookahead from x at depth 1, updating its heuristic estimates accordingly (call to function `HLRTA-LookaheadUpdate1`). Second, it selects state y of $Succ(x)$ with minimum value of $c(x,y)+h(y)$ as next state (breaking ties randomly). Third, it executes an action that passes from x to y. At this point, y is the new current state and the loop iterates. Note that the heuristic estimators computed in a trial are used as initial values in the next trial.

Function `HLRTA-LookaheadUpdate1` performs lookahead from x at depth 1. If it happens that search moves back to a state $v \in Succ(x)$ through the same arc it moved forward (that is, $d(v)=x$), then $h_2(v)$ is admissible and $H(v)$ takes it. Otherwise, $H(v)$ takes $h_1(v)$. Then, $h_1(x)$ and $h_2(x)$ are updated accordingly. If $h_1(x)$ changes, the function returns $true$, otherwise it returns $false$.

```
procedure HLRTA*($X, A, c, s, G$)
  for each $x \in X$ do $h_1(x) \leftarrow h_0(x)$; $h_2(x) \leftarrow 0$; $d(x) \leftarrow null$;
  repeat
    HLRTA-trial($X, A, c, s, G$);
    until $h_1$ does not change;

procedure HLRTA-trial($X, A, c, s, G$)
  $x \leftarrow s$;
  while $x \notin G$ do
    $dummy \leftarrow$ HLRTA-LookaheadUpdate1($x$);
    $y \leftarrow \text{argmin}_{w \in Succ(x)}[c(x, w) + H(w)]$;
    execute($a \in A$ such that $a = (x, y)$);
    $d(x) \leftarrow y$; $x \leftarrow y$;

function HLRTA-LookaheadUpdate1($x$): boolean;
  for each $v \in Succ(x)$ do
    if $d(v) = x$ then $H(v) = h_2(v)$;
    else $H(v) = h_1(v)$;
  $y \leftarrow \text{argmin}_{v \in Succ(x)}[c(x, v) + H(v)]$;
  $z \leftarrow \text{arg 2nd min}_{v \in Succ(x)}[c(x, v) + H(v)]$;
  if $h_2(x) < c(x, z) + H(z)$ then $h_2(x) \leftarrow c(x, z) + H(z)$;
  if $h_1(x) < c(x, y) + H(y)$ then $h_1(x) \leftarrow c(x, y) + H(y)$; return $true$;
  else return $false$;
```

Fig. 1. The HLRTA* algorithm

In addition to the mentioned RTA*, LRTA* and HLRTA*, there are more algorithms for real-time search. The weighted and bounded versions of LRTA* [10]; FALCONS [3], that uses the classical heuristic $g(x) + h(x)$; eFALCONS [4], a hybrid between HLRTA* and FALCONS; a new version of LRTA* [8]; and $\gamma-$Trap [1], an algorithm that controls the exploration vs. exploitation trade-off.

3 Bounded Propagation

As search progresses, heuristic values of visited states are updated. Using the information obtained at the lookahead phase, we can better estimate the cost of reaching a goal from a visited state, and this new information is stored in the heuristic value of that state. This is a general strategy in real-time search algorithms. So far, most algorithms limit heuristic updating to the current state. Recently, we have proposed to propagate consistently the change of heuristic estimate of the current state up to k states, not necessarily distinct. We call this idea *bounded propagation*, since changes are *propagated* up to a *bound* or limit of k states per step. The propagation occurs on the successors of the state that changes its heuristic estimate. If one of these successors changes, this is again propagated on its own successors. This process is iterated with a limit of k considered states. Propagation improves the heuristic quality while keeping it admissible, so search will find better heuristic estimates in the future states that will help find a solution sooner.

We have applied bounded propagation to LRTA*, producing the LRTA*(k) algorithm (in fact, LRTA* is just a particular case of LRTA*(k) with $k = 1$). LRTA*(k) performance improves greatly with respect to LRTA*, in terms of first solution quality, convergence and solution stability [6]. 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. Precise results depends on the parameter k (the higher k the more benefits at the cost of longer planning steps) and the specific problem considered. It is important to mention that these benefits are asymptotically limited.

Let x be the current state, and let us assume that $h(x)$ changes. Its new value is $h(x) = min_{v \in Succ(x)}[c(x, v) + h(v)]$. Some steps later, if it happens that $h(w)$ changes, $w \in Succ(x)$, then $h(x)$ might change again, so x should be reconsidered. But if w is not the state with $min_{v \in Succ(x)}[c(x, v) + h(v)]$, no matter the change in $h(w)$, $h(x)$ will not change, because the minimum of its successors has not changed its heuristic. That state is called a support for $h(x)$.

Formally, we say that state y is *support* of $h(x)$, denoted $y = supp(x)$, iff $y = argmin_{v \in Succ(x)}\ [c(x, v) + h(v)]$. The previous paragraph describes a simple property of bounded propagation: if state y changes its heuristic estimate, only those states x successors of y such that y is their support could change its heuristic estimate. A successor state z not supported by y will not change: z is supported by other state and as far this state does not change its heuristic, z will not change. Bounded propagation can benefit from this property, by propagating those states which are supported by the state that has changed its heuristic. The use of supports requires a table that records for each expanded state its corresponding support.

Since propagation is limited up to k states, it is meaningful to consider which states are the most adequate to be updated. Originally, we decided to limit propagation to states already expanded in the current trial. This and other alternatives are discussed in [5].

4 HLRTA*(k)

HLRTA*(k) is the algorithm that combines HLRTA* with bounded propagation. The rationale for this combination is as follows. First, it is reasonable to expect that HLRTA* would benefit from bounded propagation in the same way that LRTA* does, improving first solution quality, convergence and solution stability. Second, it has been observed experimentally that HLRTA* convergence is slower than LRTA* [4]. Since bounded propagation improves convergence, its combination with HLRTA* could be quite beneficial. Experimental results, reported in Section 5, clearly justify these assumptions.

The HLRTA*(k) algorithm appears in Figure 2. The main differences with HLRTA* appear in the `HLRTA-Trial` procedure: `HLRTA-LookaheadUpdate1` is replaced by `HLRTA-LookaheadUpdateK`. This procedure performs the updating of the current state plus bounded propagation. HLRTA* updating involves the first and second minima of heuristic values among successors of the current state, while propagation considers the first minimum only. These strategies are considered in `HLRTA-LookaheadUpdate2min` and `HLRTA-LookaheadUpdate1min`. This last function returns *true* when h_1 has changed as consequence of propagation, $false$ otherwise.

Bounded propagation is done in the `HLRTA-LookaheadUpdateK` procedure. If x is the current state of search, it initializes queue Q with capacity for k elements. After performing the 2nd minima update of x, it enters the following loop. While Q is not empty, it extracts the first state v from Q and performs 1st minima updating. If this updating has caused some change in $h_1(v)$, it has to be propagated, so the successors of v are entered in Q, provided the following conditions: there is room ($cont > 0$), a successor w has some chance to modify its heuristic estimator ($v = supp(w)$), and w belongs to the path of expanded states in the current or previous executions. As final remark, we stress the point that $d(x)$, the state that search moves from the current state, gets value *null* before the updating process. This is required to avoid that an old value of $d(x)$ could cause an erroneous propagation of heuristic estimates before it takes the next state to which search moves on.

It is not difficult to see that HLRTA*(k) is a complete algorithm. Theorem 2 of [11] on the completeness of HLRTA* remains valid, since stored heuristic values increase with the length of the path. To prove convergence to optimal paths it is required to assure that h_1 remains admissible after bounded propagation, with the following lemma.

Lemma 1(from [2], modified)**.** Let $x \in X - G$, h_1, h_2 and H as in HLRTA*. If $h_1(x) \leftarrow max(h_1(x), min_{v \in Succ(x)}(c(x,v) + H(v))$ then $h_1(x) \leq h^*(x)$.

Proof. If $h_1(x) \geq min_{v \in Succ(x)}(c(x,v) + H(v))$ there is nothing to prove. Otherwise, there is an optimal path from x to a goal that passes through a successor w. Then, $h^*(x) = c(x,w) + h^*(w) \geq c(x,w) + H(w)$. To see this, remember that $H(w)$ may be $h_1(w)$ (in which case $H(w)$ is obviously admissible) or $h_2(w)$. In general, $h_2(w)$ is not admissible, except through the path from x to w, provided that last time w was visited search moved to x as next state. In that particular case $h_2(w)$ is admissible [11]. But this is the case in which $H(w)$ can take value $h_2(w)$, when $d(w) = x$. Therefore, $H(w)$ is admissible through the path from x to w, so we can write $h^*(x) = c(x,w) + h^*(w) \geq c(x,w) + H(w)$. In particular, this is true for the minimum $c(x,v) + H(v)$ among successors of x, that is, $h^*(x) \geq min_{v \in Succ(x)}(c(x,v) + H(v))$. So $h_1(x) \leq h^*(x)$.

With this result, the proof of HLRTA* convergence to optimal paths (Theorem 5 of [11]) is also valid for HLRTA*(k), since it requires admissibility of h_1 stored heuristic. Therefore, HLRTA*(k) is a correct and complete algorithm that converges to optimal paths over repeated trials on the same problem instance.

```
procedure HLRTA*(k)$(X, A, c, s, G)$
  for each $x \in X$ do $h_1(x) \leftarrow h_0(x)$; $h_2(x) \leftarrow 0$; $supp(x) \leftarrow null$;
  $path \leftarrow \langle s \rangle$;
  repeat
    HLRTA(k)-trial$(X, A, c, s, G, k)$;
    until $h_1$ does not change;

procedure HLRTA(k)-trial$(X, A, c, s, G, k)$
  $x \leftarrow s$;
  while $x \notin G$ do
    $d(x) \leftarrow null$;
    HLRTA-LookaheadUpdateK$(x, k, path)$;
    $y \leftarrow \mathrm{argmin}_{w \in Succ(x)}[c(x,w) + H(w)]$;
    execute($a \in A$ such that $a = (x, y)$);
    $path \leftarrow$ add-last$(path, y)$;
    $d(x) \leftarrow y$;
    $x \leftarrow y$;

procedure LookaheadUpdateK$(x, k, path)$
  $Q \leftarrow \langle x \rangle$;
  $cont \leftarrow k - 1$;
  HLRTA-LookaheadUpdate2min$(x)$;
  while $Q \neq \emptyset$ do
    $v \leftarrow$ extract-first$(Q)$;
    if HLRTA-LookaheadUpdate1min$(v)$ then
      for each $w \in Succ(v)$ do
        if $w \in path \wedge cont > 0 \wedge v = supp(w)$ then
          $Q \leftarrow$ add-last$(Q, w)$;
          $cont \leftarrow cont - 1$;

procedure HLRTA-LookaheadUpdate2min$(x)$
  for each $v \in Succ(x)$ do
    if $d(v) = x$ then $H(v) = max\{h_1(v), h_2(v)\}$;
    else $H(v) = h_1(v)$;
  $z \leftarrow \arg$ 2nd $\min_{v \in Succ(x)}[c(x,v) + H(v)]$;
  if $h_2(x) < c(x,z) + H(z)$ then $h_2(x) \leftarrow c(x,z) + H(z)$;

function HLRTA-LookaheadUpdate1min$(x)$: boolean;
  for each $v \in Succ(x)$ do
    if $d(v) = x$ then $H(v) = max\{h_1(v), h_2(v)\}$;
    else $H(v) = h_1(v)$;
  $y \leftarrow \mathrm{argmin}_{v \in Succ(x)}[c(x,v) + H(v)]$;
  $supp(x) \leftarrow y$;
  if $h_1(x) < c(x,y) + H(y)$ then $h_1(x) \leftarrow c(x,y) + H(y)$; return $true$;
  else return $false$;
```

Fig. 2. The HLRTA*(k) algorithm

5 Experimental Results

We compare the performance of HLRTA*(k), for different values of k, with RTA* (first trial only), HLRTA*, LRTA*, LRTA*(k) and FALCONS (first trial, convergence and stability). In HLRTA*(k) and LRTA*(k) we maintain the table of supports and heuristics values between trials [5]. As benchmarks we use these four-connected grids where an agent can move one cell north, south, east or west: Grid35, grids of size 301×301 with a 35% of obstacles placed randomly. In this type of grid heuristics tend to be only slightly misleading. Grid70, grids of size 301×301 with a 70% obstacles placed randomly. In this type of grid heuristics could be misleading. Maze, acyclic mazes of size 181×181 whose corridor structure was generated with depth-first search. Here heuristics could be very misleading.

Results are averaged over 1000 different instances. In grids of size 301×301 the start and goal state are chosen randomly assuring that there is a path from the start to the goal. In mazes, the start is (0,0), and the goal is (180,180). As initial heuristic we use the Manhattan distance. Results for HLRTA*(k), RTA*, HLRTA* and FALCONS appear in Table 1. For LRTA* and LRTA*(k) results appear in Table 2. The results are presented in terms of solution cost ($\times 10^3$), number of expanded states ($\times 10^3$) and time per step ($\times 10^{-6}$ seconds), for the first trial; convergence (trials $\times 10^3$ to converge); and stability indexes [10].

In the first trial, the effect of propagation is better in HLRTA*(k) than in LRTA*(k). For low values of k, the solution cost obtained with HLRTA*(k) is better than the solutions obtained by LRTA*(k) and the others algorithms for all benchmarks (except for Grid35 and $k = 6$). For HLRTA*(k) in Grid35 and Grid70, as the value of k increases the cost of the solution improves, but in Maze the solution cost is similar for all values of k. Therefore, in Maze it is better to use small values of k, since high values of k means more computation per step. For high values of k the cost of solution obtained with LRTA*(k) and HLRTA*(k) are similar. The smallest solution cost is obtained by HLRTA*(k) and LRTA*(k) with $k = \infty$. Comparing with RTA*, HLRTA*(k) algorithm finds better solutions from $k = 6$ on in Grid35, and $k = 1$ on in Grid70 and Maze. Comparing with FALCONS, HLRTA*(k) always produces better solutions in Grid35, Grid70 and Maze.

Considering convergence, HLRTA*(k) obtains optimal solutions with less cost than HLRTA*, LRTA* and FALCONS for all the values of k tested on the three benchmarks. The effect of propagation is better in HLRTA*(k) than in LRTA*(k); thus with low values of k, HLRTA*(k) obtains better results than LRTA*(k). With high values of k, the solution cost obtained with LRTA*(k) and HLRTA*(k) are similar. Solution cost decreases monotonically as k increases. We observe that the worse the heuristic information is, the better HLRTA*(k) behaves with respect to its competitors (results are better in Maze than in Grid70, and in Grid70 than in Grid35). Considering trials to convergence, HLRTA*(k) requires substantially less trials than HLRTA*. The number of trials decreases steadily as k increases, from approximately a third of the trials with $k = 6$. Comparing with FALCONS, it performs better for high values of k in Grid35,

Table 1. Results for the first trial (left), convergence (middle) and stability (right) for HLRTA*(k), HLRTA*, RTA* and FALCONS. Average over 1000 instances.

				Grid35								
k	**Cost%**	**Mem.%**	**T/Step%**	**Cost%**	**Trials%**	**Mem.%**	**T/Step%**	**IAE**	**ISE**	**ITAE**	**ITSE**	**SOD**
100%	46.8	4.6	0.79	6202.2	5.0	45.1	0.8	$\times 10^6$	$\times 10^9$	$\times 10^8$	$\times 10^{11}$	$\times 10^5$
1 (HLRTA*)	100%	100%	100%	100%	100%	100%	100%	3.8	20.1	54.5	75.5	12.1
6	37%	87%	157%	39%	34%	100%	146%	1.6	6.1	8.4	13.3	5.0
15	24%	85%	193%	25%	20%	100%	186%	1.0	3.6	3.3	5.6	3.2
500	21%	123%	358%	5%	3%	99%	477%	0.3	1.1	0.1	0.2	0.7
∞	20%	126%	480%	1.3%	0.6%	93%	1355%	0.07	0.5	0.006	0.02	0.2
RTA*	66%	69%	39%	-	-	-	-	-	-	-	-	-
FALCONS	2058%	191%	54%	65%	13%	244%	55%	3.7	3898.1	6.3	88.3	10.6
				Grid70								
100%	40.0	1.5	0.75	669.8	0.6	1.66	0.77	$\times 10^3$	$\times 10^6$	$\times 10^4$	$\times 10^7$	$\times 10^2$
1 (HLRTA*)	100%	100%	100%	100%	100%	100%	100%	125.4	2037.1	719.4	1598.1	512.2
6	51%	102%	167%	29%	25%	100%	126%	58.9	498.0	127.5	188.9	172.2
15	38%	103%	211%	18%	14%	100%	159%	43.6	289.4	62.9	92.4	118.8
500	10%	102%	574%	4%	2%	100%	450%	10.1	25.0	4.3	5.9	28.4
∞	7.4%	107%	3157%	1.0%	0.6%	100%	1723.3%	2.3	4.7	0.3	0.5	0.2
RTA*	106%	100%	43%	-	-	-	-	-	-	-	-	-
FALCONS	194%	102%	47%	96%	28%	104%	49%	480.9	17268.5	4088.3	46665.2	1244.5
				Maze								
100%	11.4	7.2	0.71	11227.2	3.2	15.4	0.69	$\times 10^5$	$\times 10^9$	$\times 10^6$	$\times 10^{1}0$	$\times 10^4$
1 (HLRTA*)	100%	100%	100%	100%	100%	100%	100%	0.3	0.3	0.3	0.2	1.3
6	102%	101%	159%	18%	18%	100%	113%	0.25	0.3	0.07	0.07	0.9
15	99%	99%	212%	8%	7%	101%	122%	0.25	0.4	0.05	0.07	1.0
500	98%	98%	745%	1%	0.4%	101%	717%	0.25	0.4	0.04	0.08	1.1
∞	100%	100%	1644%	0.3%	0.08%	101%	11506%	0.25	0.4	0.04	0.08	1.0
RTA*	309%	100%	47%	-	-	-	-	-	-	-	-	-
FALCONS	772%	147%	52%	193%	25%	120%	52%	189.4	1136.2	7072.6	43695.6	680.8

Table 2. Results for the first trial (left), convergence (middle) and stability (right) for LRTA* and LRTA*(k). Average over 1000 instances.

				Grid35								
k	**Cost%**	**Mem.%**	**T/Step%**	**Cost%**	**Trials%**	**Mem.%**	**T/Step%**	**IAE**	**ISE**	**ITAE**	**ITSE**	**SOD**
100%	46.8	4.6	0.79	6202.2	5.0	45.1	0.8	$\times 10^6$	$\times 10^9$	$\times 10^8$	$\times 10^{11}$	$\times 10^5$
1 (LRTA*)	147%	104%	43%	105%	51%	100%	45%	5.2	56.5	38.1	130.9	15.9
6	35%	75%	87%	48%	34%	100%	82%	2.2	9.3	11.7	24.5	6.5
15	26%	79%	113%	29%	21%	99%	104%	1.3	5.1	4.4	8.8	3.9
500	21%	79%	249%	5%	3%	100%	308%	0.3	1.3	0.1	0.3	0.8
∞	20%	132%	336%	1.3%	0.6%	93%	998%	0.07	0.5	0.006	0.02	0.2
				Grid70								
100%	40.0	1.5	0.75	669.8	0.6	1.66	0.77	$\times 10^3$	$\times 10^6$	$\times 10^4$	$\times 10^7$	$\times 10^2$
1 (LRTA*)	366%	99%	45%	118%	52%	100%	48%	509.1	24227.7	2029.2	9145.2	1174.8
6	125%	98%	76%	53%	29%	100%	75%	200.0	3110.0	600.0	1520.0	533.0
15	67%	100%	100%	31%	17%	100%	93%	100.0	1110.0	231.0	501.0	327.0
500	13%	103%	317%	5%	3%	100%	266%	20.0	64.0	8.1	16.4	51.2
∞	7%	107%	2093%	1%	0.6%	100%	1296%	2.0	4.7	0.3	0.5	0.2
				Maze								
100%	11.4	7.2	0.71	11227.2	3.2	15.4	0.69	$\times 10^5$	$\times 10^9$	$\times 10^6$	$\times 10^{1}0$	$\times 10^4$
1 (LRTA*)	5145%	114%	49%	247%	50%	90%	49%	221.6	5338.3	6952.1	120297.1	1331.1
6	1498%	106%	83%	101%	24%	96%	89%	87.3	1100.0	1580.0	15000.0	531.0
15	819%	102%	113%	57%	12%	99%	117%	50.5	485.0	556.0	4010.0	301.8
500	241%	101%	445%	9%	2%	100%	434%	8.0	43.1	14.3	57.6	42.6
∞	102%	101%	3743%	0.4%	0.1%	101%	10128%	0.3	0.4	0.05	0.08	1.1

and gets better results in Grid70 and Maze for all values of k. Comparing with LRTA*, it perform better for all values of k. Comparing with LRTA*(k) it performs equal or better for all values of k. The good results of HLRTA*(k) come at the cost of extra computation, that is, longer planning time. It is worth noticing that low values of k generate large improvements in solution cost and number of trials, with a limited effect in planning time per step. For instance, with $k = 6$, the total cost to converge to optimal path is divided by a factor of approximately 3, the number of trials is divided by a factor of approximately 4, at the cost of increasing the time per step by a factor around 1.3.

To measure solution stability we computed the indices IAE, ISE, ITAE, ITSE, and SOD [10]. Smaller values mean better stability of solutions. HLRTA*(k) outperforms HLRTA*, LRTA* and LRTA*(k) for all k values tested in all indices for the three benchmarks. For $k = \infty$ LRTA*(k) is similar to HLRTA*(k). Something similar happens when comparing with FALCONS, except for index ITAE on Grid35 for $k = 6$, where FALCONS obtains better results.

6 Conclusions

As in the case of LRTA*, bounded propagation of heuristic changes is quite beneficial when applied to HLRTA*, improving first solution, convergence and solution stability, at the extra cost of longer planning steps. Experimentally, for $k > 1$, HLRTA*(k) requires less trials than LRTA*(k) to converge to optimal paths, when the contrary happens for $k = 1$.

Acknowledgments

Authors thank anonymous reviewers for their detailed reviews and constructive comments.

References

1. V. Bulitko. Learning for adaptive real-time search. *The Computing Research Rep. (CoRR): cs.DC/0407017*, 2004.
2. S. Edelkamp and J. Eckerle. New strategies in learning real time heuristic search. In *Proc. AAAI Workshop on On-Line Search*, pages 30–35, 1997.
3. D. Furcy and S. Koenig. Speeding up the convergence of real-time search. In *Proc. AAAI*, pages 891–897, 2000.
4. 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.
5. C. Hernandez and P. Meseguer. Improving convergence of lrta*(k). In *Proc. IJCAI Workshop on Planning and Learning in a Priori Unknown or Dynamic Domains*, pages 69–75, 2005.
6. C. Hernandez and P. Meseguer. Lrta*(k). In *Proc. IJCAI*, pages 1238–1243, 2005.
7. K. Knight. Are many reactive agents better than a few deliberative ones? In *Proc. 13th IJCAI*, pages 432–437, 1993.
8. S. Koenig. A comparison of fast search methods for real-time situated agents. In *Proc. 3rd AAMAS*, pages 864–871, 2004.
9. R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.
10. M. Shimbo and T. Ishida. Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, 146(1):1–41, 2003.
11. P. E. Thorpe. A hybrid learning real-time search algorithm. Master's thesis, Computer Science Dep., UCLA, 1994.