LRTA*(k) *

Carlos Hernández and **Pedro Meseguer**
Institut d'Investigació en Intel.ligència Artificial, CSIC
Campus UAB, 08193 Bellaterra, Spain
{chernan,pedro}@iiia.csic.es

Abstract

LRTA* is a real-time heuristic search algorithm widely used. In each iteration it updates the heuristic estimate of the current state. Here we present LRTA*(k), a new LRTA*-based algorithm that is able to update the heuristic estimates of up to k states, not necessarily distinct. Based on bounded propagation, this updating strategy maintains heuristic admissibility, so the new algorithm keeps the good theoretical properties of LRTA*. Experimentally, we show that LRTA*(k) produces better solutions in the first trial and converges faster when compared with other state-of-the-art algorithms on benchmarks for real-time search.

1 Introduction

LRTA* [Korf, 1990] is a real-time heuristic search algorithm that interleaves planning and action execution in an on-line manner. This allows LRTA* to face search problems on unknown or changing environments, tasks that are not easily performed by off-line search. This algorithm works on a search space where every state x has a heuristic estimate $h(x)$ of the cost from x to a goal. It is complete under a set of reasonable assumptions. In addition, LRTA* improves its performance over successive trials on the same problem, by recording better heuristic estimates. If $h(x)$ is admissible, after a number of trials $h(x)$ converges to their exact values along every optimal path. At this point, only optimal paths are traversed by LRTA*. Before convergence improvement is not monotonic: after a near optimal path LRTA* could find a worse path [Shimbo and Ishida, 2003].

In this paper we present LRTA*(k), an algorithm based on LRTA* with its same structure. The only difference lies in the updating strategy. LRTA* updates the heuristic estimate of a single state per iteration. LRTA*(k) updates the heuristic estimate of up to k, not necessarily distinct, states per iteration following a bounded propagation strategy. This updating maintains heuristic admissibility, so LRTA*(k) converges to exact heuristic values in optimal paths, just like LRTA*. In fact, LRTA* is a particular case of LRTA*(k) with $k = 1$.

*Partially supported by the Spanish REPLI project TIC-2002-04470-C03-03.

LRTA*(k) propagates the cost discovered by lookahead up to k states. Because of that, better heuristic estimates are propagated, causing the following benefits:

- First solution. The quality of the first solution is important for real-time search algorithms. If the first solution would involve no cycles, LRTA*(k) would behave like LRTA*. However, this rarely happens. When LRTA*(k) revisits a state, it finds a better heuristic estimate than the one LRTA* would have found, selecting a better action. Experimentally, LRTA*(k) finds shorter solutions in less computation time than LRTA*.
- Convergence. LRTA*(k) records heuristic estimates that are closer to their exact values than those recorded by LRTA*. This causes LRTA*(k) to converge faster than LRTA* (in number of steps, in number of trials and also in total CPU time) in the benchmarks tested. This also happens for other algorithms (FALCONS).
- Solution stability. Getting generally better quality solutions causes less differences between solutions and increases stability. This was not initially pursued (we were not bounding how bad a solution could be) but it has appeared as a side-effect of improving solution quality.

These benefits come at the cost of extra computation per planning step, required for bounded propagation. Limiting computation up to k updates bounds this extra cost, which could be adjusted (varying k) to the requirements of each application. In our experiments, the time per planning step has remained reasonable when compared with LRTA*.

The structure of the paper is as follows. In Section 2 we summarize related approaches. In Section 3 we present the basic elements of LRTA*. In Section 4 we describe LRTA*(k) and its new updating strategy, using the concepts of bounded propagation and support. We provide experimental results in Section 5, showing the benefits of our approach on classical real-time search benchmarks. Finally, Section 6 contains some conclusions and lines for further research.

2 Related Work

In his seminal work, Korf proposed LRTA* and RTA* (an algorithm with a different updating strategy, able to find better solutions in the first trial but without converging to optimal routes) [Korf, 1990]. After that, several approaches have

been made to to improve LRTA* on the quality of the first solution, convergence and stability. HLRTA* [Thorpe, 1994] is a hybrid between RTA* and LRTA*. It finds better solutions than LRTA* in the first trial and converges to optimal paths. As RTA*, it avoids the ping-pong effect [Edelkamp and Eckerle, 1997] but requires more memory. The weighted and bounded versions of LRTA* [Shimbo and Ishida, 2003] speed up convergence and improve solution stability, but sacrifice optimality. FALCONS [Furcy and Koenig, 2000] accelerates convergence and keeps optimality by using $g(x) + h(x)$ as heuristic function, where $g(x)$ is the cost from the start state to x. FALCONS improves convergence but it may perform a large amount of exploration in earlier trials. eFALCONS [Furcy and Koenig, 2001] is a hybrid between HLRTA* and FALCONS. It converges as FALCONS, performing a smaller amount of actions in earlier trials, although it may be greater than LRTA*. A new version of LRTA* [Koenig, 2004] improves convergence by increasing lookahead depth, causing to increase the planning time per step.$\gamma-$Trap [Bulitko, 2004] uses adaptive lookahead depth and offers control on the exploration vs. exploitation trade-off, but sacrifices optimality. Other approaches consider search with moving target [Ishida and Korf, 1991] and cooperative agents [Knight, 1993], [Goldenberg *et al.*, 2003].

3 LRTA*

We assume a state space defined by the tuple (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, that is, 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 called the heuristic estimate of x. The exact cost $h^*(x)$ is the minimum cost to go from x to a goal. If $\forall x \in X$, $h(x) \leq h^*(x)$ then h is admissible. A path $(x_0, x_1, \ldots, x_n)$ such that $h(x_i) = h^*(x_i)$, $0 \leq i \leq n$ is called optimal.

The LRTA* algorithm with lookahead at depth 1 (the case considered in this paper) and converging to optimal paths (with h admissible) appears in Figure 1. Like in [Korf, 1990], 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 `LRTA*` initializes the heuristic estimate of every state using the function h_0, and repeats the execution of `LRTA*-trial` until convergence (h does not change). At this point, an optimal path has been found. Procedure `LRTA*-trial` initializes the current state x with the start s. Then, the following loop is executed until a goal is found. First, function `LookaheadUpdate1` is executed ignoring its result. It

```
procedure LRTA*(X, A, c, s, G)
  for each x ∈ X do h(x) ← h0(x);          /* initialization */
  repeat
    LRTA*-trial(X, A, c, s, G);
    until h does not change;

procedure LRTA*-trial(X, A, c, s, G)
  x ← s;
  while x ∉ G do
    dummy ← LookaheadUpdate1(x);           /* look+upt */
    y ← argmin_{w∈Succ(x)}[c(x, w) + h(w)]; /* next state */
    execute(a ∈ A such that a = (x, y)); /* action exec */
    x ← y;

function LookaheadUpdate1(x): boolean;
  y ← argmin_{v∈Succ(x)}[c(x, v) + h(v)];  /* lookahead */
  if h(x) < c(x, y) + h(y) then
    h(x) ← c(x, y) + h(y); return true;
  else return false;
```

Figure 1: The LRTA* algorithm.

performs lookahead from x at depth 1, updating its heuristic estimate accordingly. Second, the state y of $Succ(x)$ with minimum value of $c(x, y) + h(y)$ is selected as next state (breaking ties randomly). Third, an action that passes from x to y is executed. After it, y is the new current state and the loop iterates.

Function `LookaheadUpdate1` performs lookahead from x at depth 1, and updates $h(x)$ if it is lower than the minimum cost of moving from x to one of its successors y plus its heuristic estimate $h(y)$. If $h(x)$ changes it returns $true$ otherwise returns $false$.

In a state space like the one assumed here (finite, positive costs, finite heuristic estimates) where from every state there is a path to a goal it has been proved that LRTA* is complete. In addition, if h is admissible, over repeated trials the heuristic estimates eventually converge to their exact values along every optimal path [Korf, 1990].

4 LTRA*(k)

4.1 Propagation of Changes of Heuristic Estimates

LRTA*(k) differs from LRTA* in the *propagation* of changes of heuristic estimates, which is explained in the following. For ease of presentation, first we describe unbounded propagation, and second we present bounded propagation, which is the one used in LRTA*(k).

Unbounded propagation works as follows. If it happens that the current state heuristic is updated by the effect of lookahead, LRTA* selects the next state and executes the action that moves to that state, but it does not perform further updates. The current iteration stops and future iterations will be in charge of propagating that change (if needed). Unbounded propagation propagates this change to the successors of the current state. Each successor is taken as the basis for a new lookahead, and its heuristic is updated accordingly. If this propagation causes further changes, the successors of the states whose heuristic have changed will be considered for further propagation, and so forth. The process iterates until no further changes are performed.

To illustrate unbounded propagation, let us consider the example of a 4×4 grid that appear in Figure 2, where m is the start state and p is the goal. Initial heuristic estimates are computed by Manhattan distance, and obstacles are represented by "-". From state m, LRTA* performs lookahead, updates $h(m)$ to 5 (assuming that costs to move to successor states are equal to 1), and moves to i, terminating the current iteration. Unbounded propagation performs further propagation of the $h(m)$ change. The successor of m is i, from which lookahead is performed and $h(i)$ is updated to 6. Again, this change causes to reconsider the successors of i, e and m. $h(e)$ is updated to 7, which causes to reconsider its successors a and i. $h(m)$ is updated to 7, which causes to reconsider its successor i. a is reconsidered but its heuristic estimate does not change. $h(i)$ is updated to 8, which causes to reconsider its successors e and m. $h(e)$ does not change. $h(m)$ is updated to 9, which causes to reconsider its successor i. $h(i)$ does not change. Unbounded propagation stops. This propagation presents some drawbacks for real-time search,

1. Move in bounded time [Koenig, 2001][Koenig, 2004]. The number of states involved in unbounded propagation may differ in consecutive steps, so the computational effort required may change between steps. This is against the requirement that real-time search must perform individual moves in bounded (appr. constant) time.
2. Acting on vicinity. Unbounded propagation can act very far from the current state. This violates a basic assumption of real-time search, that lookahead and update operations can only be done in the vicinity of the current state [Shimbo and Ishida, 2003].

To solve these drawbacks, we propose to limit unbounded propagation. This new version, called *bounded* propagation, limits to k the maximum number of states that can be updated in one step. In this way, the computational effort devoted to propagation is bounded, and individual moves are performed within that bound. This is acceptable as far as the agent could perform the extra computation required between successive actions without disturbing its activity.

In addition, we require another condition: updating is limited to previously expanded states. In other words, only states belonging to the path from the start to the current state can be updated. This condition tries to prevent that heuristic updating to be scattered over the search space. Combined with the condition of bounded propagation, the vicinity for updating is limited to k previously visited states.

To illustrate bounded propagation updating expanded states, the example of Figure 2 is executed with $k = 5$. State m is updated and $h(m)$ changes to 5. Since there are no previous states in the path, no propagation is made. In this iteration, 1 state (m) has been considered for update. The agent moves to i, which is updated, $h(i)$ changes to 6 and state m is reconsidered (the only successor in the path). m is updated and $h(m)$ changes to 7. This causes to reconsider state i, also in the path, but this causes no changes. In this iteration, 3 states (i, m, i) have been considered for update. The agent moves to e, which is updated, $h(e)$ changes to 7 and state i is reconsidered. $h(i)$ changes to 8, and its successors in the path, e and m are reconsidered. e causes no changes. m is

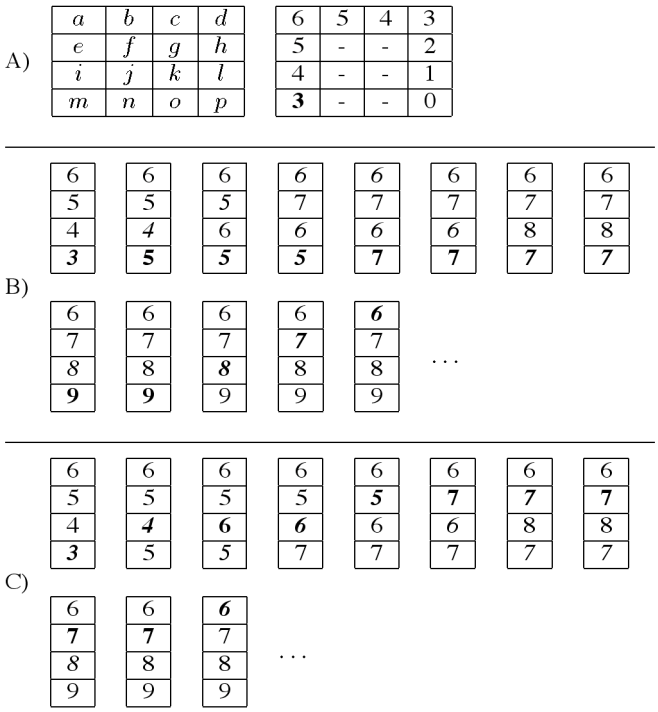

Figure 2: A 4×4 grid. (A) States (m is the start and p is the goal) and initial heuristic estimates (Manhattan distance). Obstacles are represented by "-". A number in **bold** indicates the heuristic estimate of the state where the agent is located, while numbers in *italics* represent states whose heuristic estimates should be reconsidered. (B) LRTA* with unbounded propagation in the leftmost column, until the agent reaches state a. (C) LRTA* with bounded propagation ($k = 5$) and updating previously visited states in the leftmost column, until the agent reaches state a.

updated, $h(m)$ changes to 9 and causes its successor i in the path to be reconsidered. i causes no change. In this iteration, 5 states (e, i, e, m, i) have been considered for update. At this point, agent moves to a and continues towards the goal p.

Bounded propagation depends on k, the maximum number of previous states that can be updated. We believe that bounded propagation is a suitable strategy for real-time search. This is confirmed by the experiments of Section 5 on different problems with several k values.

4.2 The Algorithm

The LRTA*(k) algorithm appears in Figure 3. Procedure `LRTA*(k)-trial` is very similar to procedure `LRTA*-trial` of Figure 1. They only differ in two points: `LRTA*(k)-trial` records the sequence of expanded states in $path$, and it executes the procedure `LookaheadUpdateK` instead of the function `LookaheadUpdate1`. Procedure `LookaheadUpdateK` performs bounded propagation as follows. It maintains a sequence Q of states candidates to update their heuristic estimates. Q is initialized with the current state. At most, k states will be entered in Q. This is controlled by the counter $cont$, initialized to $k-1$. Then, the following loop is executed until

```
procedure LRTA*(k)(X, A, c, s, G, k)
  for each x ∈ X do h(x) ← h_0(x);
  repeat
    LRTA*(k)-trial(X, A, c, s, G, k);
    until h does not change;

procedure LRTA*(k)-trial(X, A, c, s, G, k)
  x ← s; path ← ⟨s⟩;
  while x ∉ G do
    LookaheadUpdateK(x, k, path);
    y ← argmin_{w∈Succ(x)}[c(x, w) + h(w)];
    execute(a ∈ A such that a = (x, y));
    path ← add-last(path, y); x ← y;

procedure LookaheadUpdateK(x, k, path)
  Q ← ⟨x⟩; cont ← k − 1;
  while Q ≠ ∅ do
    v ← extract-first(Q);
    if LookaheadUpdate1(v) then
      for each w ∈ Succ(v) do
        if w ∈ path ∧ cont > 0 then
          Q ← add-last(Q, w); cont ← cont − 1;

function LookaheadUpdate1(x): boolean;
  y ← argmin_{v∈Succ(x)}[c(x, v) + h(v)];
  if h(x) < c(x, y) + h(y) then
    h(x) ← c(x, y) + h(y); return true;
  else return false;
```

Figure 3: The LRTA*(k) algorithm.

Q contains no states. The first state v in Q is extracted, from which lookahead is performed and it is updated accordingly. If $h(v)$ changes (`LookaheadUpdate1` of Figure 1 returns $true$), this is propagated over its successors as follows. Any w successor of v that belongs to $path$ enters Q in the last position, provided that there is still room in Q (the limit of k states has not been exhausted during the current execution of the procedure). If $h(v)$ does not change, the loop iterates processing the next state of Q. The admissibility of h after bounded propagation is guaranteed by the following lemma.

Lemma 1 *(Lemma 2 of [Edelkamp and Eckerle, 1997]). Let $x \in X - G$ and h admissible. Updating $h(x) \leftarrow max(h(x), min_{v \in Succ(x)}(c(x, v) + h(v))$ implies that $h(x) \leq h^*(x)$.*

Proof. If $h(x) \geq min_{v \in Succ(x)}(c(x, v) + h(v))$ there is nothing to prove. Otherwise, there is an optimal path from x to a goal that passes through a successor w. Then, $h^*(x) = c(x, w) + h^*(w) \geq c(x, w) + h(w)$. 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 $h(x) \leq h^*(x)$.

From this result, convergence of LRTA*(k) to optimal paths is guaranteed in the same terms as LRTA*, because Theorem 3 of [Korf, 1990] is also valid for LRTA*(k). The new algorithm inherits the good properties of LRTA*.

4.3 Supports

Algorithm LRTA*(k) can easily be improved using the notion of support of an heuristic estimate. Imagine the situation depicted in Figure 5, where each state has four successors (north, south, east, west) and the cost to move to any

```
procedure LRTA*(k)-trial(X, A, c, s, G, k)
  for each x ∈ X do supp(x) ← null;
  x ← s; path ← ⟨s⟩;
  while x ∉ G do
    LookaheadUpdateK(x, k, path);
    y ← argmin_{w∈Succ(x)}[c(x, w) + h(w)];
    execute(a ∈ A such that a = (x, y));
    path ← add-last(path, y); x ← y;

procedure LookaheadUpdateK(x, k, path)
  Q ← ⟨x⟩; cont ← k − 1;
  while Q ≠ ∅ do
    v ← extract-first(Q);
    if LookaheadUpdate1(v) then
      for each w ∈ Succ(v) do
        if w ∈ path ∧ cont > 0 ∧ v = supp(w) then
          Q ← add-last(Q, w); cont ← cont − 1;

function LookaheadUpdate1(x): boolean;
  y ← argmin_{v∈Succ(x)}[c(x, v) + h(v)]; supp(x) ← y;
  if h(x) < c(x, y) + h(y) then
    h(x) ← c(x, y) + h(y); return true;
  else return false;
```

Figure 4: Adding supports to LRTA*(k).

of them is 1. e is the current state and the path is the sequence $(\ldots, b, c, e)$. If it happens that $h(e)$ changes from 3 to 5, LRTA*$(k > 1)$ will reconsider the heuristic estimate of state c. However, no matter the change of $h(e)$, $h(c)$ will not change because the minimum of its successors, state b, has not changed its heuristic estimate. The value of $h(b)$ justifies the current value of $h(c)$. State b is called a support for $h(c)$.

Formally, state y is *support* of $h(x)$, written $y = sup(x)$, iff $y = argmin_{v \in Succ(x)}(c(x, v) + h(v))$. The previous example illustrates 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 estimate, z will not change.

LRTA*(k) can benefit from this property, by entering in Q only those states which are supported by the state that has changed its heuristic estimate. This requires minor modifications that appear in Figure 5. In `LookaheadUpdate1`, each time a state is expanded its support is recorded. In `LookaheadUpdateK`, only those states supported by the state who has changed are considered for inclusion in Q. The use of supports requires a table that records for each expanded state its corresponding support. This duplicates the memory requirements of LRTA*.

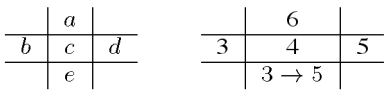

Figure 5: A state space (left) and its heuristic estimates (right). The path followed by the agent is $(\ldots, b, c, e)$, e is the current state and $h(e)$ changes from 3 to 5. State c does not change because its support (state b) has not changed.

Grid35			
k	Cost%	States%	Time/Step%
	68755.6=100%	7014.3=100%	0.00034=100%
1 (LRTA*)	100%	100%	100%
6	24%	89%	181%
15	17%	100%	237%
51	15%	117%	324%
500	**14%**	133%	523%
∞	**14%**	86%	754%
RTA*	45%	**66%**	**91%**
FALCONS	1402%	183%	126%
Grid70			
k	Cost%	States%	Time/Step%
	146382.3=100%	1447.1=100%	0.00034=100%
1 (LRTA*)	100%	100%	100%
6	34%	**99%**	169%
15	18%	100%	222%
51	9%	102%	328%
500	4%	104%	708%
∞	**2%**	108%	4668%
RTA*	29%	101%	**97%**
FALCONS	53%	103%	105%
Maze			
k	Cost%	States%	Time/Step%
	588001.0=100%	8218.7=100%	0.00035=100%
1 (LRTA*)	100%	100%	100%
6	29%	93%	171%
15	16%	90%	231%
51	8%	**88%**	387%
500	5%	89%	914%
∞	**2%**	89%	7683%
RTA*	6%	**88%**	**96%**
FALCONS	15%	129%	106%

Table 1: Results for the first trial on Grid35, Grid70 and Maze benchmarks, averaged over 1000 instances.

Grid35				
k	Cost%	Trials%	MaxStates%	Time/Step%
	6493803.3=100%	2561.4=100%	7014.3=100%	0.00036=100%
1 (LRTA*)	100%	100%	100%	**100%**
6	48%	63%	**89%**	163%
15	36%	48%	100%	189%
51	29%	43%	117%	205%
500	26%	**42%**	133%	216%
∞	**25%**	**42%**	136%	223%
FALCONS	62%	26%	245%	122%
Grid70				
k	Cost%	Trials%	MaxStates%	Time/Step%
	790030.4=100%	306.8=100%	1480.7=100%	0.00037=100%
1 (LRTA*)	100%	100%	**100%**	**100%**
6	44%	57%	**100%**	148%
15	27%	37%	103%	179%
51	18%	26%	105%	219%
500	10%	17%	107%	302%
∞	**2%**	**5%**	111%	906%
FALCONS	81%	55%	104%	103%
Maze				
k	Cost%	Trials%	MaxStates%	Time/Step%
	27767147.9=100%	1584.3=100%	11360.7=100%	0.00035=100%
1 (LRTA*)	100%	100%	**100%**	**100%**
6	39%	48%	120%	169%
15	23%	24%	128%	232%
51	11%	11%	128%	360%
500	5%	7%	120%	663%
∞	**2%**	**6%**	116%	1792%
FALCONS	78%	50%	134%	105%

Table 2: Results for convergence on Grid35, Grid70 and Maze benchmarks, averaged over 1000 instances.

5 Experimental Results

We compare the performance of LRTA*(k), for different values of k, with RTA* (first trial only), LRTA* and FALCONS (first trial, convergence and stability). As benchmarks we use four-connected grids where an agent can move one cell north, south, east or west. We use the following benchmarks:

1. Grid35. Grids of size 301x301 with a 35% of obstacles placed randomly. In this type of grid heuristics tend to be only slightly misleading.
2. Grid70. Grids of size 301x301 with a 70% obstacles placed randomly. In this type of grid heuristics could be misleading.
3. Maze. Acyclic mazes of size 181x181 whose corridor structure was generated with depth-first search. In this type of grid heuristics could be very misleading.

In each case, results are averaged over 1000 different instances. In grids of size 301x301 the start and goal state are chosen randomly with the restriction that there is a path from the start state to the goal state. In mazes, the start state is (0,0), and the goal state is (180,180). As initial heuristic between two states we use the Manhattan distance.

Table 1 contains the results for the first trial, in terms of solution cost, number of expanded states and time per step (in milliseconds). The smallest solution cost is obtained by LRTA*(k) with $k = \infty$, that is, propagation runs without limit of states, updating states which are in the path connecting the start with the current state. Comparing with LRTA*, our algorithm produces better solutions for all values of k tested. Because of bounded propagation, LRTA*(k) requires more computation than LRTA* per step. This can be adjusted using the k parameter. If the agent has enough planning time per step, it can use a high k because this largely increases the solution quality. Otherwise, low values of k improve solution quality and differences with LRTA* in planning time per step remain reasonable. With k between 6 and 15, LRTA*(k) uses approximately twice the planning time per step required by LRTA*, decreasing solution cost by a factor between 3 and 5. Comparing with RTA*, our algorithm finds better solutions from $k = 6$ on in Grid35, $k = 15$ on in Grid70 and $k = 500$ on in Maze. Comparing with FALCONS, our algorithm always produces better solutions in Grid35 and Grid70. In Maze, it works better from $k = 51$ on.

We compare the learning process of LRTA*(k) with LRTA* and FALCONS (RTA* is excluded, it does not converge to optimal routes). Table 2 contains the results for convergence to optimal paths, in terms of total cost, number of trials, maximum number of expanded states and planning time per step. Considering solution cost, LRTA*(k) obtains better results than LRTA* and FALCONS for the values of k tested on the three benchmarks. Solution cost decreases monotonically as k increases. We observe that the worse the heuristic information is, the better LRTA*(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, LRTA*(k) requires many fewer trials than LRTA*. The number of trials decreases steadily as k increases, from approximately half of the trials with $k = 6$. Comparing with FALCONS, it does not perform better for Grid35, but gets better results in Grid70 and Maze. The good results of LRTA*(k) come at the cost of extra computation,

Grid35					
k	IAE $\times 10^6$	ISE $\times 10^9$	ITAE $\times 10^8$	ITSE $\times 10^{11}$	SOD $\times 10^5$
1 (LRTA*)	5.2	56.5	38.1	130.9	15.9
6	2.3	10.6	11.7	26.3	6.8
15	1.7	7.6	6.5	14.2	5.1
51	1.3	5.8	4.5	8.6	4.1
500	1.2	4.3	3.9	6.5	3.6
∞	1.1	3.7	3.8	6.2	3.4
FALCONS	3.7	3898.1	6.3	88.3	10.6
Grid70					
k	IAE $\times 10^3$	ISE $\times 10^6$	ITAE $\times 10^4$	ITSE $\times 10^7$	SOD $\times 10^2$
1 (LRTA*)	509.1	24227.7	2029.2	9145.2	1174.8
6	188.4	2931.4	702.6	1265.5	592.0
15	112.9	950.8	301.7	501.4	379.7
51	67.0	327.5	119.7	166.0	210.8
500	26.5	75.2	25.0	26.8	65.3
∞	4.0	5.9	2.7	1.1	5.8
FALCONS	480.9	17268.5	4088.3	46665.2	1244.5
Maze					
k	IAE $\times 10^5$	ISE $\times 10^9$	ITAE $\times 10^6$	ITSE $\times 10^{10}$	SOD $\times 10^4$
1 (LRTA*)	221.6	5338.3	6952.1	120297.1	1331.1
6	82.0	403.8	2109.0	4490.6	412.4
15	50.7	200.3	659.2	1242.9	188.8
51	25.1	93.4	157.0	340.7	104.0
500	9.6	28.4	34.8	66.6	48.6
∞	1.5	1.6	4.2	1.6	7.0
FALCONS	189.4	1136.2	7072.6	43695.6	680.8

Table 3: Results for solution stability on Grid35, Grid70 and Maze benchmarks, averaged over 1000 instances.

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 a bit higher than 2, the number of trials is divided by a factor of appr. 2, at the cost of increasing the time per step by a factor around 1.6.

In these algorithms the number of expanded states determines the amount of memory used. LRTA*(k) records for each expanded state its heuristic estimate and its support, while LRTA* records its heuristic estimate only. This means that LRTA*(k) requires twice the memory used by LRTA* per expanded state. FALCONS records for each expanded state the values of g and h, so it has similar requirements as LRTA*(k) per expanded state. On the benchmarks tested, all the algorithms expand a relatively close number of states (except FALCONS in Grid35, that requires a larger amount). Roughly speaking, we can say that LRTA*(k) requires from two to two and a half times the memory required by LRTA* on the same instances, and a similar (and in some cases larger) amount of the memory is required by FALCONS.

To measure solution stability we computed the indices IAE, ISE, ITAE, ITSE, and SOD [Shimbo and Ishida, 2003]. IAE provides the sum of the error in the convergence. ISE provides the square of the sum of the error in the convergence, it penalizes large overshoots. ITAE and ITSE are two time-weighted versions of IAE and ISE, that impose large penalties on sustained errors. SOD sums up the difference in solution costs between two consecutive trials when the solution worsens. If SOD is equal to 0 convergence is monotonic.

Table 3 contains the stability indices for LRTA*(k), LRTA* and FALCONS for each grid. LRTA*(k) outperforms LRTA* for all k values tested in all indices for the three benchmarks. Something similar happens when comparing with FALCONS, except for index ITAE on Grid35 for $k = 6, 15$, where FALCONS obtains better results. In Grid35 FALCONS have some advantage with respect to ITAE because it performs a great amount of moves in earlier trials (see Tables 1 and 2), which are less penalized by the index. Solution stability improves monotonically as k increases. Again, results of Table 3 suggest that LRTA*(k) is more robust than its competitors in presence of misleading heuristics.

6 Conclusions

LRTA*(k) is a real-time search algorithm that converges to optimal routes. Based on LRTA*, it performs bounded propagation of heuristic changes up to k states, which causes longer planning steps. Experimentally, LRTA*(k) shows a substantial performance improvement with respect to LRTA* and FALCONS, in terms of first trial, convergence and solution stability. Improvements depend on k: the higher k, the better results at the cost of longer planning steps. Small k values cause moderate planning times but large benefits. LRTA*(k) can be very useful when the agent has enough time between actions to accommodate the planning time required. Future research includes the setting of k and the combination with other heuristic techniques.

References

[Bulitko, 2004] V. Bulitko. Learning for adaptive real-time search. *The Computing Research Rep. (CoRR): cs.DC/0407017*, 2004.

[Edelkamp and Eckerle, 1997] 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.

[Furcy and Koenig, 2000] D. Furcy and S. Koenig. Speeding up the convergence of real-time search. In *Proc. AAAI*, pages 891–897, 2000.

[Furcy and Koenig, 2001] 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.

[Goldenberg *et al.*, 2003] M. Goldenberg, A. Kovarksy, X. Wu, and J. Schaeffer. Multiple agents moving target search. In *Proc. 18th IJCAI*, pages 1536–1538, 2003.

[Ishida and Korf, 1991] T. Ishida and R. E. Korf. Moving target search. In *Proc 12th IJCAI*, pages 204–211, 1991.

[Knight, 1993] K. Knight. Are many reactive agents better than a few deliberative ones? In *Proc. 13th IJCAI*, pages 432–437, 1993.

[Koenig, 2001] S. Koenig. Agent-centered search. *Artificial Intelligence Magazine*, 22(4):109–131, 2001.

[Koenig, 2004] S. Koenig. A comparison of fast search methods for real-time situated agents. In *Proc. 3rd AAMAS*, pages 864–871, 2004.

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

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

[Thorpe, 1994] P. E. Thorpe. A hybrid learning real-time search algorithm. Master's thesis, Computer Science Dep., UCLA, 1994.