Improving LRTA*(k)

Carlos Hernández[1] 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 identify some weak points of the LRTA*(k) algorithm in the propagation of heuristic changes. To solve them, we present a new algorithm, LRTA*$_{LS}$(k), that is based on the selection and updating of the interior states of a local space around the current state. It keeps the good theoretical properties of LRTA*(k), while improving substantially its performance. It is related with a lookahead depth greater than 1. We provide experimental evidence of the benefits of the new algorithm on real-time benchmarks with respect to existing approaches.

1 Introduction

Real-time search interleaves planning and action execution in an on-line manner. This allows to face search problems in domains with incomplete information, where it is impossible to perform the classical search approach.

A real-time agent has a limited time to planify the next action to perform. In such limited time, it can explore only a relatively small part of the search space that is around its current state. This part is often called its local space.

This paper considers some weak points of the updating mechanism of the LRTA*(k) algorithm [Hernandez and Meseguer, 2005]. With this purpose, we present LRTA*$_{LS}$(k), based on local space selection and updating. Local space selection is learning-oriented. Local space is divided into interior and frontier. Updating is done from the frontier into the interior states, so some kind of lookahead is done. Each state is updated only once. If the initial heuristic is admissible, it remains admissible after updating. LRTA*$_{LS}$(k) keeps the good properties of LRTA*(k).

The paper structure is as follows. In Section 2 we expose the basic concepts used. In Section 3 we describe some weak points of LRTA*(k). In Section 4, we present LRTA*$_{LS}$(k), explaining the lookahead and update mechanisms. In Section 5, we relate LRTA*$_{LS}$(k) with lookahead greater than 1. In Section 6, we provide experimental results. Finally, in Section 7 we extract some conclusions.

[1] On leave from Universidad Católica de la Santísima Concepción, Caupolicán 491, Concepción, Chile.

2 Preliminaries

The state space is defined as (X, A, c, s, G), where (X,A) is a finite graph, $c : A \rightarrow [0,\infty)$ 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 $A \subset X \times X \setminus \{(x, x)|x \in X\}$ 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 $c(x, y) = c(y, x)$. $k(n, m)$ is the cost of the path between state n and m. The successors of a state x are $Succ(x) = \{y|(x, y) \in A\}$. A heuristic function $h : X \rightarrow [0,\infty)$ associates with each state x an approximation $h(x)$ of the cost of a path from x to a goal, where $h(g)$=0, $g \in G$. The exact cost $h^*(x)$ is the minimum cost to go from x to a 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.

LRTA* [Korf 1990] is a real-time search algorithm that is complete under a set of reasonable assumptions. 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. In each step, LRTA* updates the heuristic estimation of the current state to make it consistent with the values of their successors [Bonnet and Geffner, 2006]. Originally, LRTA* performs one update per step. Some versions perform more than one update per step.

3 LRTA*(k)

LRTA*(k) [Hernandez and Meseguer, 2005], is a LRTA*-based algorithm that consistently propagates heuristic changes up to k states per step, following a bounded propagation strategy. If the heuristic of the current state x changes, the heuristic of its successors may change as well, so they are reconsidered for updating. If the heuristic of some successor v changes, the successors of v are reconsidered again, and so on. The propagation mechanism is bounded to reconsider up to k states per step. This updating strategy maintains heuristic admissibility, so LRTA*(k) inherits the good properties of LRTA* (in fact, LRTA*(k reduces to LRTA* when k=1). Experimentally, LRTA*(k)

improves significantly with respect to LRTA*.

The updating mechanism is implemented with a queue Q, that maintains those states that should be reconsidered for updating. We have identified some weak points:

1. If y enters Q, 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 updating per state?
3. The order in which states enter Q, combined with the value of k parameter, may affect the final result.

As example, in the Figure 1 (*i*), a, b, c and d are states, the left column is the number of the update iteration, and the numbers below the states are their heuristic values. Moving from a state to a successor costs 1. Sucessors are revised in left-right order. The current state is d, iteration 0 shows the initial heuristic values. In iteration 1, $h(d)$ changes to 4, $Q = \{c\}$. In iteration 2, $h(c)$ changes to 5, $Q = \{b, d\}$. In iteration 3, $h(b)$ does not change. In iteration 4, $h(d)$ changes to 6, $Q = \{c\}$. In iteration 5, $h(c)$ does not change. After five iterations no more changes are made and the process stops. We see that c and d enters two times in Q, d is updated twice while c is updated only once. If $k = 3$, the final result depends on the entering order in Q. If the left successor goes first, the states d, c, b will enter Q in that order, producing the heuristic estimates of Figure 1 (*ii*). The agent moves to c, and then it may move again to d. If the right successor goes first, the states in Q will be d, c, d, causing heuristic values equal to those of iteration 5. Then, the agent moves from d to c, and then to b, not revisiting d.

In [Hernandez and Meseguer, 2005] it is required that updates are done on previously visited states only. In that case, weak point 1 is solved by the use of supports [2]. However, there is a version of LRTA*(k), called LRTA*$_2$(k), which updates any state (not just previously visited states). In this case, supports can only partially solve this issue.

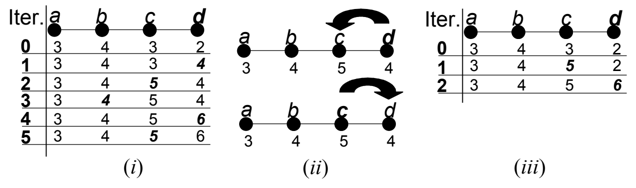

Figure 1. (*i*) Example of LRTA*(k) updating; (*ii*) if successors enter Q in left-right order, and k=3, state d can be revisited; (*iii*) example of LRTA*$_{LS}$(k) updating.

[2] There, support usage does not consider ties. It is straightforward to develop a support management including ties.

4 LRTA*$_{LS}$(k)

Trying to solve those weak points, we propose a new algorithm LRTA*$_{LS}$(k), based on the notion of local space around the current state. 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 inmediate and completely, so $I \cap F = \emptyset$. LRTA*$_{LS}$(k) selects the local space around the current state and then updates its interior states. Each state is updated only once. Under some circumstances, we can prove that every interior state will change its heuristic. The number of interior states is upper-bounded by k. If the initial heuristic is admissible, LRTA*$_{LS}$(k) maintains the admissibility, so it inherits the good properties of LRTA* (and LRTA*(k)).

4.1 Selecting the Local Space

We want to find a local space around the current state such that its interior states will have a high chance to update its heuristic. We propose the following method,

```
1. Set $I = \emptyset$, $Q = \{x \mid x$ is the current state$\}$.
2. Loop until $Q$ 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 (that is, 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 $Succ(w) \setminus I$ in $Q$.
3. The set $F$ surrounds $I$ inmediate and completely.
```

For example, let us apply this strategy to Figure 1 (*i*). It starts with $I = \{d\}$ since $h(d) < \min_{v \in Succ(d)} h(v) + 1$. Then, it checks the successors of d, adding c to I, $I = \{c, d\}$ since $h(c) < \min_{v \in \{Succ(c)-I\}} h(v) + 1$. Then, it checks the successors of c that do not belong to I, that is, b. Since $h(b) = \min_{v \in \{Succ(b)-I\}} h(v) + 1$, it is not possible to add more states to I. Then $I = \{c, d\}$ and $F = \{b\}$. If the heuristic is initially consistent, all interior states of the local space (I, F) built with the previous method will change their heuristic, as proven in the following result.

Proposition 1. *If the heuristic is initially consistent, starting from x LRTA*$_2$ ($k=\infty$) will update every state of I.*

Proof. By induction in the number of states of I. If $|I| = 1$, then $I = \{x\}$, the current state. In this case, $h(x)$ is updated because by construction it satisfies the updating condition. Let us assume the proposition for $|I| = p$, and let us prove it for $|I| = p + 1$. Let $z \in I$, $z \neq x$, $y = \text{argmin}_{v \in Succ(z)} h(v) + c(z,v)$. If $y \notin I$, z satisfies the updating condition by construction. If $y \in I$, let $I' = I - \{z\}$. Since $|I'| = p$, every state of I' will be updated, so y will be updated, passing from $h_{old}(y)$ to $h_{new}(y)$, $h_{old}(y) < h_{new}(y)$. We will see that this causes z to be updated. We consider two cases:
1. If y is still $\text{argmin}_{v \in Succ(z)} c(z,v) + h(v)$. The initial heuristic is consistent so $h(z) \leq c(z,y) + h_{old}(y) < c(z,y) + h_{new}(y)$. So z satisfies the updating condition and $h(z)$ will change.
2. If $w = \text{argmin}_{v \in Succ(z)} c(z,v) + h(v)$, $w \neq y$. If $w \notin I'$, z satisfies the updating condition by construction. If $w \in I'$, w has

passed from $h_{old}(w)$ to $h_{new}(w)$, $h_{old}(w) < h_{new}(w)$. Since y was the initial minimum, we know that $c(z,y) + h_{old}(y) < c(z,w) + h_{new}(w)$. So, $h(z) \leq c(z,y) + h_{old}(y) < c(z,w) + h_{new}(w)$. Then, z satisfies the updating condition and $h(z)$ will change.

LRTA*$_2$(k=∞) reaches all interior states because every successor of a changing state enters the queue Q, which has no limit in length (k=∞). qed.

If the heuristic is admissible but not consistent, some interior states may not change. For instance, consider the states a-b-c-d in a linear chain, with heuristic values 1, 1, 5, 6, and b is the current state. The previous method will generate $I = \{b, c\}$. $h(b)$ will change to 2, but $h(c)$ will not change.

4.2 Updating the Local Space

LRTA*$_{LS}$(k) updates the interior states of (I, F) as follows,

Loop while I is not empty:

1. Calculate the pair of connected states (i, f), $i \in I$, $f \in F$, $f \in Succ(i)$, such that $(i, f) = \text{argmin}_{v \in I, w \in F, w \in Succ(v)} c(v, w) + h(w)$.
2. Update $h(i) = c(i,f) + h(f)$.
3. Remove i from I, include i in F.

For example, in Figure 1 (i), $I = \{c, d\}$ and $F = \{b\}$, the results of each iteration with the new updating appear in Figure 1 (iii). In iteration 1, $(i, f) = (c, b)$, and $h(c)$ changes to 5. In iteration 2, $(i, f) = (d, c)$, and $h(d)$ changes to 6. As $I = \emptyset$ process stops. Update keeps admissibility, as proved next.

Proposition 2. *If the initial heuristic is admissible, after updating interior states the heuristic remains admissible.*

Proof. Let $(i, f) = \text{argmin}_{v \in I, w \in F, w \in Succ(v)} c(v, w) + h(w)$. Since $i \in I$ and connected to a frontier state f, it will be updated (it satisfies the updating condition by construction). There is an optimal path from i to a goal that passes through a successor j. We distinguish two cases:

1. $j \in F$. After updating

$$\begin{aligned} h(i) &= c(i,f) + h(f) \\ &\leq c(i,j) + h(j) && (i,f) \text{ minimum } c(v,w) + h(w) \\ &\leq c(i,j) + h^*(j) && h \text{ admissible} \\ &= h^*(i) && \text{because optimal path} \end{aligned}$$

2. $j \notin F$. If I contains no goals, the optimal path at some point will pass through F. Be p the state in the optimal path that belongs to F and q the interior state in the optimal path just before p. After updating

$$\begin{aligned} h(i) &= c(i,j) + h(j) \\ &\leq c(q,p) + h(p) && (i,f) \text{ minimum } c(v,w) + h(w) \\ &\leq k(i,p) + h(p) && k(i,p) \text{ includes } c(q,p) \\ &\leq k(i,p) + h^*(p) && h \text{ admissible} \\ &= h^*(i) && \text{because optimal path} \end{aligned}$$

In both cases $h(i) \leq h^*(i)$ so h remains admissible. qed.

Proposition 3. *Every state updated by LRTA*$_2$(k=∞) is also updated by LRTA*$_{LS}$(k=∞).*

The proof is direct, but it is omitted for space reasons. With these selection and updating strategies, the weak points of LRTA*(k) are solved partial or totally. Regarding point 1, if the heuristic is consistent all interior states will change its heuristic estimator (propositions 1 and 3). If the heuristic is just admissible, some internal states may not change (example in Section 4.1). Point 2 is totally solved, since the updating strategy considers every state only once. About point 3, it is not solved since the final result may still depend on the order in which successors are considered and the value of k parameter. Since LRTA*$_{LS}$(k) performs a better usage of k than LRTA*(k) (non repeated states and with high chance to change), we expect that, with the same k value, the former will depend less than the latter on the successor ordering. If the heuristic is consistent, after updating interior nodes the local space is locally consistent and agent decisions are locally optimal [Pemberton and Korf, 1992].

It is important to see that the new update method makes a more aggressive learning than LRTA*(k), because LRTA*(k) updates a state x considering all its successors, while LRTA*$_{LS}$(k) considers those successors that are frontier states only. One update of LRTA*$_{LS}$(k) may be greater than one update of LRTA*(k). For instance, see the updating of $h(d)$ in Figure 1 (i): it goes from 2 to 4, and then to 6, in two steps. In Figure 1 (iii), $h(d)$ goes from 2 to 6 in one step.

4.3 The algorithm

The LRTA*$_{LS}$(k) is a algorithm based on LRTA* similar to LRTA*(k). The algorithm appears in Figure 2. The main differences with LRTA* appear in the `LRTA-LS-Trial` procedure, calling to `SelectionUpdateLS`. This procedure performs the selection and updating of the local space around the current state. Procedure `SelectionUpdateLS` first performs the selection of the local space and then updates it. To select the local space, a queue Q keeps states candidates to be included in I or in F. Q is initialized with the current state and F and I are empty. At most k states will be entered in I. This is controlled by the counter *cont*. The following loop is executed until Q contains no states or *cont* is equal to k. The first state v in Q is extracted. The state $y \leftarrow \text{argmin}_{w \in Succ(v), w \notin I} h(w) + c(v, w)$ is computed. If $h(v)$ satisfies the updating condition, then v enters I, the counter increments and those successors of v that are not in I or Q enters in Q in the last position. Otherwise, v enters F. When exiting the loop, if Q still contains states then these states go to F. Once the local space has been selected, its interior states are updated as follows. The pair $(i, f) \leftarrow \text{argmin}_{v \in I, w \in F, w \in Succ(v)} h(w) + c(v, w)$ is selected, $h(i)$ is updated, i exits I and enters F.

Since the heuristic always increases, compleneteness holds (Theorem 1 of [Korf, 1990]). Heuristic admissibility is maintained, so convergence of to optimal paths is guaranteed in the same terms as LRTA* (Theorem 3 of [Korf, 1990]). LRTA*$_{LS}$(k) inherits the good properties of LRTA*.

```
procedure LRTA*-LS(k) $(X, A, c, s, G, k)$
  for each $x \in X$ do $h(x) \leftarrow h_0(x)$;
  repeat
    LRTA*-LS-trial$(X, A, c, s, G, k)$;
  until $h$ does not change;

procedure LRTA*-LS-trial$(X, A, c, s, G, k)$
  $x \leftarrow s$;
  while $x \notin G$ do
    SelectUpdateLS$(x, k)$;
    $y \leftarrow \mathrm{argmin}_{w \in Succ(x)} [h(w) + c(x,w)]$;(Break ties randomly)
    execute($a \in A$ such that $a = (x, y)$);
    $x \leftarrow y$;

procedure SelectUpdateLS $(x, k)$
  $Q \leftarrow \langle x \rangle$; $F \leftarrow \emptyset$; $I \leftarrow \emptyset$; $cont \leftarrow 0$;
  while $Q \neq \emptyset \wedge cont < k$ do
    $v \leftarrow$ extract-first$(Q)$;
    $y \leftarrow \mathrm{argmin}_{w \in Succ(v) \wedge w \notin I}\, h(w) + c(v, w)$;
    if $h(v) < h(y) + c(v, y)$ 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 $F \leftarrow F \cup Q$;
  while $I \neq \emptyset$ do
    $(i, f) \leftarrow \mathrm{argmin}_{v \in I,\, w \in F,\, w \in Succ(v)}\, h(w) + c(v, w)$;
    $h(i) \leftarrow \max[h(i), c(i,f) + h(f)]$;
    $I \leftarrow I - \{i\}$;
    $F \leftarrow F \cup \{i\}$;
```

Figure 2: LRTA*$_{LS}$(k) algorithm.

5 Lookahead and Local Space

Real-time agents restrict planning to the local space [Koenig, 2001]. Basically an agent performs three steps: (i) lookahead on the local space, (ii) updating the local space and (iii) moving in the local space.

On lookahead, in RTA*, LRTA* [Korf, 1990; Knight, 1993] and in LRTS [Bulitko and Lee, 2006] the local space is determined by the depth of the search horizon. The search tree, rooted at the current state, is explored breadth-first until depth d. In the LRTA* version of [Koenig, 2004] and in RTAA* [Koenig and Likhachev, 2006], the local space is determined by a bounded version of A*, limiting the number of states that can enter in CLOSED. LRTA*$_{LS}$(k) tries to find those states that will change in the updating process. Lookahead is limited by the k parameter: a maximum of k states can enter the set of interior states. About updating, there are two approaches: updating the current state only (like LRTS with the max-min rule), and updating all interior states of the local space (like RTAA* that update all states in CLOSED and LRTA*$_{LS}$(k)). Regarding moving, for unknown environments the free space assumption is assumed [Koenig *et al.*, 2003]. In RTAA* the agent tries to move to the frontier of the local space. If it finds an obstacle, the whole process is repeated again. In LRTA*-based algorithms the agent moves to its best successor.

Figure 3 shows an example of a goal-directed navigation task on a grid (legal moves: north, south, east, west; all moves of cost 1) with three different algorithms: LRTA*$_{LS}$(k=6), RTAA* with lookahead 6 and LRTS (with parameters γ=1 and T=∞) with lookahead 3. The visibility radius is one [Bulitko and Lee, 2006]. Black cells are obstacles. The initial heuristic is the Manhattan distance. The current state is indicated with a circle. With similar lookahead, LRTA*$_{LS}$(k) expands less states (the set of interior states contains 5 states with k = 6, because only those 5 states will changes their heuristic) and after updating it obtains a heuristic more informed than RTAA* and LRTS.

The updating mechanism of LRTA*$_{LS}$(k) is a general method. Therefore, it is possible to use it with different lookahead mechanisms, like those of RTTA* and LRTS.

6 Experimental Results

We compare LRTA*$_{LS}$(k) versus LRTA*(k). In addition, we compare LRTA*$_{LS}$(k) with RTAA* and LRTS, algorithms with lookahead greater than 1. As benchmarks we use these four-connected grids,

- Grid35: grids of size 301x301 with a 35% of obstacles placed randomly. In this type of grid heuristics tend to be slightly misleading.
- Grid70: grids of size 301x301 with a 70% of obstacles placed randomly. In this type of grid heuristics could be misleading.
- Maze: acyclic mazes of size 151x151 whose corridor structure was generated with depth-first search. Here heuristics could be very misleading.

Results are averaged over 1000 different instances. In grids and mazes, the start and goal state are chosen randomly assuring that there is a path from the start to the goal. All actions have cost 1. As initial heuristic we use the Manhattan distance. The visibility radius of the agent is 1. Results are presented in percentage with respect to the value of row 100%, in terms of total search time, solution cost, trials to convergence, number of updated states, and time per step, for the convergence to optimal path.

Comparing LRTA*$_{LS}$(k) and LRTA*(k), we consider first trial and convergence to optimal paths. Results for convergence to optimal paths appear in Table 1, where the LCM algorithm [Pemberton and Korf, 1992] is also included (LCM has the same updating mechanism as LRTA*(k), but without supports and limitation of entrances in the queue). LRTA*$_{LS}$(k) converges to optimal paths with less cost, number of trials and total time than LRTA*(k) for all values of k tested between 1 and ∞ (except for Grid35 in total time with k = 5). Respect to time per step, LRTA*$_{LS}$(k) obtains better results than LRTA*(k) except for Grid35. LRTA*$_{LS}$(k) requires slightly more memory than LRTA*(k) (except for Grid70 with $k = \infty$). LRTA*($k = \infty$) and LCM have the same cost and number of trials to convergence, but LRTA*(k=∞) improves over LCM in time because the use of supports. LRTA*$_{LS}$($k = \infty$) has similar results in cost and number of trials to convergence, requiring less time than LRTA*(k=∞) and LCM (except for Grid35 in time per step). We observe that the worse the heuristic information is, the better results LRTA*$_{LS}$(k) obtains, for all values of k. For Grid35, LRTA*$_{LS}$ (5) improves over LRTA*(5) in steps and trials, but it requires more running time. This is due to the extra overhead of LRTA*$_{LS}$(k) has over LRTA*(k) (minimizing on two vs. one argument), that counterbalances the benefits in steps and trials.

Results for first trial appear in Table 2. LRTA*$_{LS}$(k) obtains a solution with less cost than LRTA*(k) for all tested values of k between 1 and ∞ (except for Maze with k = 5). Respect to time, LRTA*$_{LS}$(k) obtains better solution for Maze and Grid70 than LRTA*(k). In Grid35, LRTA*(k) shows better results in total time and time per step. Trying to improve the running time, we have developed LRTA*$_{LS}$(k)-path, that only updates states that have been visited previously (like LRTA*(k) and LCM). LRTA*$_{LS}$(k)-path improves the cost of the solution and total time for small values of k, and this improvement is larger when worse is the heuristic information (in Grid35 it improves for k = 5 only, in Maze it improves for k = 5, 25, 61 and 113). This fact can be explained as follows. Due to the free space assumption, LRTA*$_{LS}$(k) may perform an inefficient use of k, updating states that are blocked (will never appear in any solution). Since a great part of the states in Maze are blocked, is better to update only states in the current path, as the results show. Considering memory, LRTA*$_{LS}$(k) uses more memory than LRTA*(k) and LRTA*$_{LS}$(k)-path with increasing k and more than LCM with k = ∞.

Comparing with lookahead algorithms, results for first trial appear in Table 2. Comparing LRTA*$_{LS}$(k) and RTAA*, the former obtains costs equal to or better than the latter in Grid70 and Maze for all values of k. However, from k = 25, RTAA* obtains slightly better costs in Grid35. RTAA* takes more time in finding a solution for all values of k in all benchmarks. Considering time per step, LRTA*$_{LS}$(k) requires a smaller time for all values of k in all benchmarks (except for Maze with k = 5, 25). From k = 25 in Maze and from k = 61 in G70, RTAA* uses more memory than LRTA*$_{LS}$(k). In G35, RTAA* uses less memory for all values k. LRTS uses the radius of a circle centered in the current state as lookahead parameter, while LRTA*$_{LS}$(k) and RTAA* use the number of the states they consider in advance. To perform a fair comparison, we use LRTS (with parameters γ=1 and T=∞) lookahead values that generate circles including k states, where k is the lookahead of LRTA*$_{LS}$(k) and RTAA*. LRTS does not show a good performance in terms of cost and time to find a solution respect to LRTA*$_{LS}$(k). LRTS uses less memory than LRTA*$_{LS}$(k) and smaller time per step.

In summary, LRTA*$_{LS}$(k) shows better performance than LRTA*(k) both in first trial and convergence to optimal paths. Sometimes updating only previously visited states could improve performance: if the heuristic quality is good, it is advisable to update any state, but with bad quality heuristics it is safer to update visited states only. In first trial, LRTA*$_{LS}$(k) performs equal or better than RTAA* in two benchmarks, while there is no clear winner for the third. LRTA*$_{LS}$(k) performs better than LRTS. On memory usage, LRTS wins, while the second is unclear between LRTA*$_{LS}$(k) and RTAA*.

7 Conclusions

Trying to solve some inefficiencies in LRTA*(k) propagation, we have developed LRTA*$_{LS}$(k) a new real-time search algorithm. It is based on the selection and update of a local space, composed of interior and frontier states. The number of interior states is upper-bounded by k. LRTA*$_{LS}$(k) updates the interior states from the heuristic values of frontier states, so it performs a kind of lookahead with varying depth. If the heuristic is initially consistent, all interior states are updated. Experimentally, we show that LRTA*$_{LS}$(k) improves over LRTA*(k) (lookahead 1). and improves over RTAA* and LRTS (with lookahead greater than 1).

References

[Bonnet and Geffner, 2006] Blai Bonet and Hector Geffner. Learning Depth-First Search: A Unified Approach to Heuristic Search in Deterministic and Non-Deterministic Settings, and its application to MDPs. *Proc. of the Int. Conf. Automated Planning and Scheduling.* UK. 2006.

[Bulitko and Lee, 2006] Vadim Bulitko and Greg Lee. Learning in Real Time Search: A Unifying Framework. *Journal of AI Research*, 25:119-157. 2006.

[Hernandez and Meseguer, 2005] Carlos Hernández and Pedro Meseguer. LRTA*(k). In *Proc. of the 19th Int. Joint Conference on Artificial Intelligence*, 1238–1243, Edimburgh, Scotland, Juny 2005.

[Knight, 1993] Kevin Knight. Are many reactive agents better than a few deliberative ones?. In *Proc. of the 13th Int. Joint Conference on Artificial Intelligence*, 1993.

[Koenig and Likhachev 2006] Sven Koenig and Maxim Likhachev. Real-Time Adaptive A*. In *Proc. of the Int. Joint Conference on on Autonomous Agents and Multi-Agent Systems*. 2006.

[Koenig *et al*. 2003] Sven Koenig, Craig A. Tovey and Yury V. Smirnov. Performance bounds for planning in unknown terrain. *Artificial Intelligence*. 147(1-2): 253-279, 2003.

[Koenig, 2001] Sven Koenig. Agent-Centered Search. *Artificial Intelligence Magazine*, 22, (4), 109-131, 2001.

[Koenig, 2004] Sven Koenig. A comparison of fast search methods for real-time situated agents. In *Proc. of the 3th Int. Joint Conference on on Autonomous Agents and Multi-Agent Systems*, 864-871, 2004.

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

[Pemberton and Korf, 1992] Joseph Pemberton and Richard Korf. Making locally optimal decisions on graphs with cycles *Tech. Rep. 920004.* Comp. Sc. Dep. UCLA, 1992.

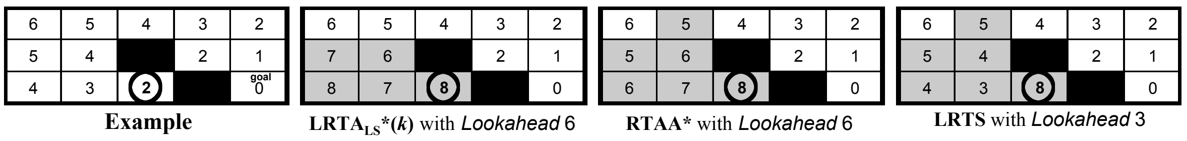

Figure 3. Example of selection and update of the local space of three algorithms with similar lookahead. The grey states are expanded and updated to the value that appears in the figure.

	Maze					Grid70					Grid35				
	LRTA*$_{LS}$(*k*)					**LRTA*$_{LS}$(*k*)**					**LRTA*$_{LS}$(*k*)**				
(a)	(b)	(c)	(d)	(e)	(f)	(b)	(c)	(d)	(e)	(f)	(b)	(c)	(d)	(e)	(f)
100%	17,309	1.67E+07	1,057	9,866	0.00103	622	634,636	304	1,655	0.00098	636	568,157	650	13,171	0.0011
1	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	44%	32%	42%	105%	137%	50%	38%	52%	101%	129%	66%	40%	56%	103%	164%
25	13%	8%	11%	116%	172%	17%	11%	16%	106%	162%	36%	14%	21%	116%	254%
61	7%	3%	5%	119%	200%	9%	5%	8%	110%	188%	30%	9%	10%	124%	341%
113	5%	2%	3%	120%	235%	7%	3%	5%	116%	227%	27%	7%	9%	130%	420%
181	4%	1%	2%	121%	278%	6%	2%	3%	123%	298%	27%	5%	7%	136%	492%
∞	7%	0.1%	0.2%	124%	5371%	9%	1.2%	1.6%	561%	782%	40%	3%	4%	249%	1252%
	LRTA*(*k*)					**LRTA*(*k*)**					**LRTA*(*k*)**				
1	86%	100%	100%	100%	86%	91%	100%	100%	100%	91%	92%	100%	100%	100%	92%
5	55%	38%	43%	106%	144%	56%	43%	53%	100%	129%	58%	44%	57%	102%	132%
25	40%	15%	15%	107%	269%	36%	17%	21%	100%	212%	40%	19%	24%	103%	213%
61	36%	9%	9%	107%	394%	31%	11%	13%	100%	293%	36%	13%	15%	105%	281%
113	34%	7%	6%	107%	513%	29%	8%	9%	100%	368%	35%	10%	12%	107%	338%
181	33%	5%	5%	107%	622%	28%	6%	8%	100%	439%	34%	9%	10%	108%	385%
∞	49%	0.1%	0.2%	107%	34281%	26%	1.2%	1.6%	100%	2223%	42%	6%	4%	115%	751%
	LCM					**LCM**					**LCM**				
∞	61%	0.1%	0.2%	107%	42745%	33%	1.2%	1.6%	100%	2813%	61%	6%	4%	115%	1083%

Table 1. Results for convergence to optimal path. (a) = lookahead or *k* value, (b) = total search time, (c) trajectory cost (total number of steps), (d) = number of trial, (e) = memory (number of updated states), (f) = search time per step.

	Maze				Grid70				Grid35			
	LRTA*$_{LS}$(*k*)				**LRTA*$_{LS}$(*k*)**				**LRTA*$_{LS}$(*k*)**			
(a)	(b)	(c)	(d)	(e)	(b)	(c)	(d)	(e)	(b)	(c)	(d)	(e)
100%	430	399,666	5,610	0.0011	104	94,011	1,476	0.0011	5	4,059	720	0.0012
1	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	47%	32%	108%	147%	55%	35%	102%	157%	75%	40%	103%	190%
25	17%	7%	120%	241%	17%	8%	107%	220%	65%	22%	126%	293%
61	18%	4%	121%	403%	10%	3%	110%	305%	74%	20%	148%	376%
113	21%	4%	122%	568%	12%	3%	114%	442%	85%	19%	169%	442%
181	24%	3%	123%	734%	16%	3%	120%	604%	93%	19%	187%	498%
∞	86%	2%	131%	3649%	35%	3%	389%	1363%	258%	19%	476%	1390%
	LRTA*(*k*)				**LRTA*(*k*)**				**LRTA*(*k*)**			
1	83%	100%	100%	83%	84%	100%	100%	84%	85%	100%	100%	85%
5	32%	21%	100%	150%	52%	37%	100%	143%	59%	42%	99%	139%
25	32%	11%	100%	306%	30%	12%	100%	252%	59%	29%	103%	203%
61	40%	9%	101%	467%	26%	8%	100%	345%	65%	28%	106%	237%
113	44%	7%	100%	609%	25%	6%	100%	426%	67%	27%	106%	253%
181	49%	7%	101%	724%	25%	5%	100%	502%	68%	26%	106%	261%
∞	346%	2%	101%	14809%	92%	3%	103%	3315%	78%	26%	107%	299%
	LCM				**LCM**				**LCM**			
∞	430%	2%	101%	18445%	118%	3%	103%	4229%	103%	26%	107%	394%
	LRTA*$_{LS}$(*k*)-Path				**LRTA*$_{LS}$(*k*)-Path**				**LRTA*$_{LS}$(*k*)-Path**			
1	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%
5	28%	18%	102%	151%	50%	31%	101%	161%	68%	38%	99%	180%
25	16%	6%	101%	276%	16%	7%	101%	234%	66%	27%	105%	246%
61	18%	4%	100%	452%	13%	4%	102%	363%	70%	26%	105%	273%
113	22%	3%	100%	623%	17%	3%	103%	576%	74%	26%	106%	286%
181	25%	3%	100%	833%	24%	3%	103%	833%	77%	26%	107%	294%
∞	114%	2%	101%	4870%	53%	3%	103%	1906%	79%	26%	107%	301%
	RTAA*				**RTAA***				**RTAA***			
1	113%	100%	100%	113%	114%	100%	100%	114%	115%	100%	100%	115%
5	52%	39%	102%	133%	61%	39%	67%	154%	94%	49%	66%	192%
25	26%	11%	148%	230%	27%	10%	86%	270%	112%	22%	73%	504%
61	38%	6%	182%	597%	36%	5%	115%	712%	208%	18%	73%	1151%
113	58%	4%	196%	1361%	59%	3%	140%	1680%	379%	17%	70%	2257%
181	85%	3%	201%	2473%	90%	3%	167%	2960%	538%	17%	72%	3255%
	LRTS				**LRTS**				**LRTS**			
1	115%	100%	100%	150%	110%	100%	100%	148%	119%	100%	100%	141%
2	92%	73%	99%	127%	84%	62%	85%	134%	106%	72%	76%	147%
4	104%	87%	105%	120%	80%	57%	77%	140%	136%	67%	59%	204%
6	107%	87%	107%	123%	87%	57%	74%	152%	199%	68%	51%	291%
8	111%	85%	107%	131%	97%	56%	73%	173%	261%	62%	44%	422%
10	125%	89%	110%	140%	112%	55%	72%	203%	342%	57%	40%	604%

Table 2. Results for the first trial. (a) = lookahead or *k* value, (b) = total search time, (c) trajectory cost (total number of steps), (d) = memory (number of updated states), (e) = search time per step.