

# Lookahead, Propagation and Moves in Real-Time Heuristic Search \*

**Carlos Hernández**

Universidad Católica Santísima Concepción  
Caupolicán 491, Concepción, Chile  
chernan@ucsc.cl

**Pedro Meseguer**

IIIA, CSIC  
Campus UAB, 08193 Bellaterra, Spain  
pedro@iiia.csic.es

## Abstract

Bounded propagation of heuristic changes has been shown beneficial for real-time search methods with lookahead = 1. Here we combine bounded propagation with lookahead > 1, keeping the cycle *lookahead, update, move* of initial LRTA\* (propagation is included in *update*). Bounded propagation can start from any heuristic inaccuracy found during lookahead, and update can go outside the local search space. After update, promising states found during lookahead could no longer be "promising", and this has to be considered for the next move. We give evidences of the benefits of this new approach, on several benchmarks with different heuristics.

## 1 Introduction

Let us consider an agent with a path-finding task from a start position to a goal in an unknown environment. It can only sense the surrounding area within the range of its sensors, and remembers previously visited positions. This may happen in real-time computer games [Bulitko and Lee, 2006] and control in robotics [Koenig, 2001]. Off-line search, like A\* [Hart *et al.*, 1968], is not appropriate because requires to know the terrain in advance. Incremental A\* methods, like D\* [Stentz, 1995] and D\*Lite [Koenig and Likhachev, 2002], and real-time (RT) search methods [Korf, 1990] allow solving this task. If the first-move delay is required to be short, incremental A\* is discarded because it requires to compute the complete solution before start moving, which may be long. RT search remains the only applicable strategy for this task.

RT search interleaves planning and action execution in an on-line manner. In the planning phase the agent plans one or several actions which are performed in the action execution phase (we assume that an action produces an elementary move from a state to one of its successors). The agent has a short time to perform the planning phase. Due to this, RT

methods restrict search to the *local search space*, a small part of the state space around the current state, whose size is independent of the state space size. Searching in this local space is called *lookahead*, where some heuristic inaccuracies may be detected, and better estimates can be computed. The set of states which may change their heuristic is the *local learning space*. The agent finds how to move in the local search space and plans one or several actions, to be done in the next action execution phase. The whole process iterates until finding a goal. At each step, RT methods compute the beginning of the trajectory from the current state to the goal, without guarantee to produce an optimal one. Some methods, after repeated executions on the same instance, converge to an optimal path.

Recently, propagation of heuristic changes has improved RT search performance, considering lookahead = 1 (immediate neighbors). In this paper, we combine bounded propagation with lookahead > 1. This extension is not trivial because (i) propagation can start from any heuristic inaccuracy detected during lookahead, while previously it started from the current state only, (ii) available repair resources have to be equitably shared among detected inaccuracies, (iii) after update, the best state found during lookahead may change its heuristic and be no longer be "the best", which has to be considered when deciding the next move. These ideas are implemented in the LRTA\*<sub>LS</sub>(*k, d*) algorithm, where parameters *k* and *d* determine the size of the local learning and the local search spaces, respectively. LRTA\*<sub>LS</sub>(*k, d*) presents a novelty: the local learning space is not necessarily included in the local search space. An unexpected result appears: propagation is more important than lookahead, in the sense that if we allow  $\delta$  extra states, we obtain better results if they are allocated to the local learning space than if they were allocated to the local search space (LRTA\*<sub>LS</sub>(*k +  $\delta$ , d*) obtains better results than LRTA\*<sub>LS</sub>(*k, d +  $\delta$* )). This is experimentally supported using different benchmarks and heuristics.

The paper structure is as follows. We define the problem and summarize some solving approaches in section 2. We present the new algorithm in section 3, including an example of its behavior. Experimental results appear in section 4. We extract some conclusions in section 5.

## 2 Preliminaries

The state space is  $(X, A, c, s, G)$ , where  $(X, A)$  is a finite graph,  $c : A \mapsto [\epsilon, \infty)$ ,  $\epsilon > 0$ , is a cost function that asso-

\*Partially supported by the projects TIN2006-15387-C03-01 and Fondecyt #11080063

	Local Search Space	Local Learning Space	Learning Rule	Control Strategy
LRTA*	$\{x\} \cup succ(x)$	$\{x\}$	$max\{h(x), \min_{v \in succ(x)} [c(x, v) + h(v)]\}$	move to best successor
LRTA*( $k$ )	$\{x\} \cup succ(x)$	$Q$	$max\{h(y), \min_{v \in succ(y)} [c(y, v) + h(v)]\}$	move to best successor
LRTA* <sub>LS</sub> ( $k$ )	$\{x\} \cup succ(x)$	$I$	$max\{h(y), \min_{f \in F} [k(y, f) + h(f)]\}$ where $k[y, f]$ is the cheapest cost from $y$ to $f$ in $I$	move to best successor
LSS-LRTA*	$OPEN \cup CLOSED$ of A* started at $x$	$CLOSED$ of A* started at $x$	$max\{h(y), \min_{f \in OPEN} [k(y, f) + h(f)]\}$ where $k[y, f]$ is the cheapest cost from $y$ to $f$ in $CLOSED$	move to best state in $OPEN$ by the cheapest path in $CLOSED$
RTAA*	$OPEN \cup CLOSED$ of A* started at $x$	$CLOSED$ of A* started at $x$	$max\{h(y), f(z) - dist(x, y)\}$ where $z$ is the best state in $OPEN$	move to best state in $OPEN$ by the cheapest path in $CLOSED$
LRTS	$\{t   dist(x, t) \leq d\}$	$\{x\}$	$max\{h(x), [k(x, w) + h(w)]\}$ , where $w$ is $argmax_{i=1, \dots, d} \min_t  dist(x, t)=i  [k(x, t) + h(t)]$	move to best state at distance $d$ by the cheapest path in LSS

Figure 1: Summary of some RT search algorithms:  $x$  is the current state,  $y$  is any state of the local learning space.

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

A RT search algorithm governs the behavior of an agent, which contains in its memory a map of the environment (memory map). Initially, the agent does not know where the obstacles are, and its memory map contains the initial heuristic. The agent senses the environment around its position, identifying obstacles within its sensors range and uploading this information in its memory map. The agent executes the *planning phase* performing (i) lookahead: local search around the current state in its memory map; (ii) learning: if better heuristic values are found, these values are backed-up, causing to change the heuristic of one or several states; (iii) action selection: one or several actions are selected for execution. The *action execution* phase consists of performing the selected action in the environment. As result, the agent moves to a new state that becomes the current state. The whole process is repeated until finding a goal. Obstacles are discovered as they fall in the agent sensor range. Following [Bulitko et al., 2007], most existing RT search algorithms can be described in terms of:

- *Local search space*: those states which are visited in the lookahead step of each planning episode.
- *Local learning space*: those states which may change its heuristic in the learning step of each planning episode.
- *Learning rule*: how to change the heuristic of a state.
- *Control strategy*: how to select the next move.

Some of them are summarized in Figure 1 (for details the reader should consult the original sources). We assume that the distance between any two neighbor states is 1. LRTA\* [Korf, 1990] is considered with lookahead 1. Its cycle

is as follows. Let  $x$  be the current state and  $y = \arg \min_{z \in Succ(x)} [c(x, z) + h(z)]$ . If  $h(x) < c(x, y) + h(y)$  (the *updating condition*), LRTA\* updates  $h(x)$  to  $c(x, y) + h(y)$ . In any case, the agent moves to the state  $y$ . With lookahead  $d > 1$ , the local search space is  $\{t | dist(x, t) \leq d\}$  (the set of states at distance lower or equal to  $d$ ) and the learning rule is the minimin strategy, while the local learning space and control selection remain unchanged. In a finite state space, with a positive heuristic of minimum and finite values, where from every state there is a path to a goal, LRTA\* is complete. If  $h$  is admissible, over repeated trials (each trial takes as input the heuristic computed in the previous trial), the heuristic converges to its exact values along every optimal path (with random tie-breaking) [Korf, 1990].

Regarding LRTA\*( $k$ ) [Hernandez and Meseguer, 2005], it combines LRTA\* (lookahead 1) with bounded propagation. If the heuristic of the current state changes, this change is propagated to its successors. This idea is recursively applied to the successors that change (and their successors, etc.). Propagation is bounded: up to a maximum of  $k$  states can be reconsidered per planning step. States to be reconsidered are introduced in a queue  $Q$ , which has capacity up to  $k$  states. The idea of bounded propagation is better implemented in LRTA\*<sub>LS</sub>( $k$ ) [Hernandez and Meseguer, 2007]. If the current state changes, the local learning space  $I, |I| \leq k$ , is constructed from all states whose heuristic may change as consequence of that change. The learning rule computes the shortest path from the  $I$  frontier.

Koenig proposed LSS-LRTA\* [Koenig, 2004] [Koenig and Sun, 2009], which performs lookahead using A\*, until the number of states in  $CLOSED$  reaches some limit. Then, the heuristic of every state in  $CLOSED$  is updated following a shortest paths (Dijkstra) algorithm from the states in  $OPEN$ . A new version called RTAA\* was proposed [Koenig and Likhachev, 2006], which is faster and simpler to implement. Its only difference is the learning rule, which updates using the shortest distance to the best state in  $OPEN$ . In both algorithms, the best state in  $OPEN$  is the next state to move.

Bulitko and Lee proposed the LRTS algorithm [Bulitko and Lee, 2006], which performs lookahead using a breadth-first strategy until reaching nodes at distance  $d$  from the current state. Then, the heuristic of the current state is updated with the maxi-min learning rule. The control strategy selects the best state at level  $d$  as the next state to move.

In unknown environments, moves are computed with the *free space assumption*: if a state is not within the agent sensor range and there is no memory of an obstacle, it is assumed

feasible. Executing moves, if an obstacle is found in a feasible state, execution stops and another planning starts.

### 3 $LRTA^*_{LS}(k, d)$

The ideas of deeper lookahead and bounded propagation are combined in the  $LRTA^*_{LS}(k, d)$  algorithm. It keeps the basic cycle of lookahead, update and move of the original  $LRTA^*$  [Korf, 1990]. Each part is described in the following.

**Lookahead.** Bounded propagation starts from a heuristic inaccuracy, where the updating condition holds (if no inaccuracy exists, no change is done and there is nothing to propagate). In previous approaches [Hernandez and Meseguer, 2005; 2007], bounded propagation considered lookahead = 1 (immediate neighbors), so propagation started from a heuristic inaccuracy detected at the current state. If lookahead > 1, all expanded states are tested for the updating condition, and several heuristic inaccuracies can be detected. In this case, update does a collective repair of all detected inaccuracies. This is the basis for the combination of lookahead and propagation. Lookahead is done by  $A^*$ , until  $CLOSED$  contains  $d$  states or a goal is selected for expansion.

**Update.** When lookahead = 1 (immediate neighbors), update is done by building a set  $I$ ,  $|I| \leq k$ , of interior states, to be updated from the set  $F$  of frontier states using a shortest paths algorithm<sup>1</sup> ( $LS$  approach [Hernandez and Meseguer, 2007]). In the case of lookahead > 1, when several inaccuracies can be detected, we also build the sets  $I$  and  $F$ . The set  $I$  of interior states is built considering each heuristic inaccuracy; because of this  $I$  may finally be the union of several unconnected subsets. The set  $F$  of frontier states surrounds  $I$  immediate<sup>2</sup> and completely. Update is done by a shortest paths (Dijkstra) algorithm. Since  $|I| \leq k$ , a key point for performance is how these up to  $k$  possible updates are distributed among the detected inaccuracies. If lookahead detects  $\{x_1, x_2, \dots, x_p\}$  inaccurate states in  $CLOSED$ , the set  $I$  is built as follows: first it includes the inaccurate states  $x_1, x_2, \dots, x_p$ ; second it includes  $x_1$  successors satisfying the updating condition, the same with  $x_2$  to  $x_p$  successors (in this order); third, it includes the successors of successors of  $x_1$  to  $x_p$  that entered  $I$  and satisfy the updating condition; etc. This process iterates until no more successors satisfy the updating condition or  $I$  reaches the limit of  $k$  states. This strategy is implemented in the function `SelectLS` of Figure 3, which contains a queue  $Q$  of states to be considered and the set  $I$  to be built.  $Q$  is initialized with  $\{x_1, x_2, \dots, x_p\}$ . Successors of a state  $y$  that satisfies the updating condition which are not in  $Q \cup I$ , enter  $Q$  by rear. The set  $I$  is adaptively built from the inaccurate states  $x_1, x_2, \dots, x_p$ , some  $x_1$  successors, some  $x_2$  successors, ..., some  $x_p$  successors, some successors of  $x_1$  successors, etc. until the upper limit of  $k$  states is reached or no more states satisfy the conditions to enter  $I$ .

**Move.** When lookahead = 1, the move is done from the current state to the best immediate successor after update. With lookahead > 1,  $first(OPEN)$  is the most promising

state after lookahead. Update changes (increases) the heuristic of some states in  $CLOSED$  (some states out of  $CLOSED$  could also change). So after update,  $first(OPEN)$  may no longer be the most promising. Let  $OPEN'$  be the list of states in  $OPEN$  ordered by the heuristic function  $f = g + h$  with the new  $h$  value computed by update, and let  $x$  be the current state. We propose the following moving rule: if  $h(x) \geq h(first(OPEN'))$  (that is, if  $first(OPEN')$  is not less promising than  $x$ ), the agent moves to  $first(OPEN')$ ; otherwise, the agent moves to the best successor of  $x$ .

In summary,  $LRTA^*_{LS}(k, d)$  works as follows,

- Local search space. Following [Koenig, 2004], lookahead is done using  $A^*$  [Hart *et al.*, 1968]. It stops when  $|CLOSED| = d$  or a goal is expanded. The local search space is formed by  $CLOSED \cup OPEN$ .
- Local learning space. If the heuristic of several states  $x_1, x_2, \dots$  in  $CLOSED$  change, the local learning space  $I$  (and its  $F$ ) are computed as explained above.  $|I| \leq k$ .
- Learning rule. Propagation of heuristic changes in  $I$  is done using the Dijkstra shortest paths algorithm, as done in [Koenig, 2004].
- Control strategy.  $OPEN'$  is composed by the states of  $OPEN$  ordered by  $f = g + h$  with the new  $h$  computed by the learning rule, and  $x$  is the current state. If  $h(x) \geq h(first(OPEN'))$ , the agent moves to  $first(OPEN')$ . Otherwise, the agent moves to  $x$  best successor.

We stress that, in  $LRTA^*_{LS}(k, d)$ , the local search and local learning spaces are *independently* developed: the local search space is developed by  $A^*$  search, while the local learning space is the set  $I$ , built as explained above. Parameters  $d$  and  $k$  determine the size of local search and local learning spaces, respectively. As novelty, the local learning space is not necessarily included in the local search space (in fact, this is true for all algorithms including bounded propagation, check Figure 1). The local learning space was totally included in the local search space of other RT search algorithms (the local learning space was either the current state  $x$  or the set  $CLOSED$  when performing lookahead with  $A^*$ ). A graphical depiction of this idea appears in Figure 2. This feature allows  $LRTA^*_{LS}(k, d)$  to focus on heuristic depressions which could go beyond the limits of the local search space. We think that this is a crucial reason for the reported performance.

The  $LRTA^*_{LS}(k, d)$  algorithm appears in Figure 3. The central procedure is `LRTA*-LS(k, d)-trial`, that is executed once per trial until finding a solution (**while** loop, line



Figure 2: Relationship between the local search space (dark grey) and the local learning space (light grey). Previous approaches without propagation (left). With propagation as the current approach (right).

<sup>1</sup>The update procedure of [Hernandez and Meseguer, 2007] is a shortest paths algorithm, although it was not presented in this form.

<sup>2</sup>By "immediate" we mean that a state in  $F$  is not in  $I$  but it is successor of some state in  $I$ .

```

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

```

```

procedure LRTA*-LS( $k, d$ )-trial( $X, A, c, s, G, k, d$ )
1  $x \leftarrow s$ ;
2 while  $x \notin G$  do
3    $changes \leftarrow A^*(x, d, G)$ ;  $OPEN' \leftarrow OPEN$ ;
4   if  $changes \neq \emptyset$  then
5      $(I, F) \leftarrow SelectLS(changes, k)$ ;  $Dijkstra(I, F)$ ;
6     reorder  $OPEN'$  with the new  $h$ ;
7   if  $h(x) \geq h(\text{first}(OPEN'))$  then
8      $path \leftarrow \text{compute path from } x \text{ to first}(OPEN')$ ;
9   else  $path \leftarrow \text{best-successor}(x)$ ;
10  while  $path \neq \emptyset$  do
11     $y \leftarrow \text{extract-first}(path)$ ;
12    execute( $a \in A$  such that  $a = (x, y)$ );  $x \leftarrow y$ ;

```

```

function SelectLS( $changes, k$ ): pair of sets;
1  $Q \leftarrow changes$ ;  $I \leftarrow \emptyset$ ;
2 while  $|I| < k \wedge Q \neq \emptyset$  do
3    $y \leftarrow \text{extract-first}(Q)$ ;
4    $z \leftarrow \text{argmin}_{w \in Succ(y) \wedge w \notin I} [c(y, w) + h(w)]$ ;
5   if  $h(y) < c(y, z) + h(z)$  then
6      $I \leftarrow I \cup \{y\}$ ;
7   for each  $w \in Succ(y) - (I \cup Q)$  do add-last( $Q, w$ );
8 compute  $F$  such that it surrounds  $I$  immediate and completely;
9 return ( $I, F$ );

```

Figure 3: The  $LRTA^*_{LS}(k, d)$  algorithm.

2). This procedure works as follows. First, it performs lookahead from the current state  $x$  by  $A^*$  search (line 3).  $A^*$  executes until (i) there are  $d$  states in  $CLOSED$  or (ii) it expands a goal state. In any case, it returns  $changes$  that is the set of expanded states that satisfy the updating condition. If this set is not empty, these changes are done and propagated: the local learning space is built and updated using the shortest paths algorithm (line 5). The  $OPEN'$  list (initialized with  $OPEN$ ) is ordered by  $f$  with the new  $h$  (line 6). If the first state of  $OPEN'$  is at least as promising as  $x$  (line 7),  $path$  contains the state sequence connecting  $x$  with it (line 8); otherwise  $path$  contains  $x$  best successor (line 9). Actions passing from one state to the next in  $path$  are done (lines 10-12).

Since the heuristic always increases,  $LRTA^*_{LS}(k, d)$  is complete (Theorem 1 [Korf, 1990]). If the heuristic is initially admissible, updating the local learning space with shortest paths algorithm keeps admissibility [Koenig, 2004], so  $LRTA^*_{LS}(k, d)$  converges to optimal paths.  $LRTA^*_{LS}(k, d)$  inherits the good properties of  $LRTA^*$  about completeness and convergence to optimal paths. As one might expect,  $LRTA^*_{LS}(k, d)$  collapses into  $LRTA^*(k)$  when  $d = 1$ .

An example how  $LRTA^*_{LS}(k, d)$  works appears in Figure 4. It is a 4-connected grid with obstacles (black cells) with the initial Manhattan heuristic. To break ties among states with same  $f$  we prefer states with greater  $g$ . Successors of a state with the same  $h$  value are considered in the order east, south, west, north. Each move costs 1. The small circle is the agent.

In the first row, there is the behavior of LSS-LRTA\*. In (1), there is the initial situation. In (2), after  $A^*$

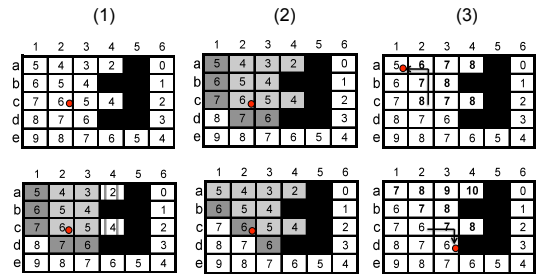


Figure 4: A 4-connected grid with Manhattan heuristic. First row, LSS-LRTA\* with  $d = 8$ : (1) initial state, (2) after lookahead with  $A^*$  ( $CLOSED$  in light grey,  $OPEN$  in dark grey), (3) after update (new values in bold) the agent moves. Second row,  $LRTA^*_{LS}(k = 8, d = 8)$ : (1) after lookahead with  $A^*$ , heuristic inaccuracies appear with vertical bars. In (2), the sets  $I$  (light grey) and  $F$  (dark grey) are shown. In (3), after updating  $I$  from  $F$  (new values in bold) the agent moves.

lookahead with  $d = 8$  ( $A^*$  executes until there are 8 states in  $CLOSED$ ), light grey cells are states in  $CLOSED = \{c2, c3, c4, b3, a3, a4, b2, a2\}$  and dark grey cells are states in  $OPEN = \{a1, d3, b1, d2, c1\}$  (ordered by  $f$ ). The local learning space coincides with  $CLOSED$ . In (3), we have the new heuristic values of states in  $CLOSED$  from  $OPEN$  using the shortest paths algorithm. The agent moves to  $a1$ , the first state of  $OPEN$ .

In the second row, there is the behavior of  $LRTA^*_{LS}(k, d)$  with  $d = 8$  and  $k = 8$ . Lookahead by  $A^*$  produces the same  $OPEN$  and  $CLOSED$  lists. In (1) we observe the two inaccuracies detected: states  $a4$  and  $c4$ . Function  $SelectLS(\{c4, a4\}, 8)$  computes sets  $I$  and  $F$  as follows,

$$\begin{aligned}
 Q &= \{c4, a4\} & I &= \{\} \\
 Q &= \{a4, c3\} & I &= \{c4\} \\
 Q &= \{c3, a3\} & I &= \{c4, a4\} \\
 Q &= \{a3\} & I &= \{c4, a4\} \\
 Q &= \{b3, a2\} & I &= \{c4, a4, a3\} \\
 Q &= \{a2, c3, b2\} & I &= \{c4, a4, a3, b3\} \\
 Q &= \{c3, b2, a1\} & I &= \{c4, a4, a3, b3, a2\} \\
 Q &= \{b2, a1, d3, c2\} & I &= \{c4, a4, a3, b3, a2, c3\} \\
 Q &= \{a1, d3, c2, b1\} & I &= \{c4, a4, a3, b3, a2, c3, b2\} \\
 Q &= \{d3, c2, b1\} & I &= \{c4, a4, a3, b3, a2, c3, b2, a1\}
 \end{aligned}$$

so the sets  $F = \{d3, c2, b1\}$  and  $I = \{c4, a4, a3, b3, a2, c3, b2, a1\}$  (in (2), in dark grey and light grey, respectively). In (3), we have the new heuristic values of states in  $I$  from the shortest paths algorithm. The new list  $OPEN'$  orders states in  $OPEN$  using the new heuristic function, so  $OPEN' = \{d3, b1, d2, c1, a1\}$ . The agent moves to  $d3$ , the first state of  $OPEN'$  (notice that, after update,  $f(a1) = 3 + 7$  while  $f(d3) = 2 + 6$ ).

In Figure 4 we observe similarities and differences between  $LRTA^*(k, d)$  and LSS-LRTA\*. Both have the same local search space developed by  $A^*$ . They differ in the local learning space,  $I$  for the former and  $CLOSED$  for the latter: observe the differences in light grey states in column (2). Both have the same updating algorithm, shortest paths. As result of having different local learning spaces, updated heuristics

Strong heuristic	Maze cost ( $\times 10^3$ )							Grid35 cost ( $\times 10$ )							Baldur cost ( $\times 10^2$ )						
	$d$	1	5	10	20	40	80	120	1	5	10	20	40	80	120	1	5	10	20	40	80
LSS-LRTA*	409	120	71	45	29	20	<b>15</b>	406	139	106	85	74	69	68	182	54	37	24	16	10	8
RTAA*	409	145	80	50	31	21	16	406	186	127	95	81	73	70	182	86	52	31	20	11	9
LRTA* $_{LS}(5, d)$	134	105						163	131						59	51					
LRTA* $_{LS}(10, d)$	62	54	61					118	103	98					34	36	29				
LRTA* $_{LS}(20, d)$	36	34	36	39				94	88	82	81				18	13	13	15			
LRTA* $_{LS}(40, d)$	22	23	26	26	27			85	81	77	73	72			9	8	7	8	8		
LRTA* $_{LS}(80, d)$	17	16	19	19	18	19		80	77	73	70	70	69		6	5	5	5	5	5	5
LRTA* $_{LS}(120, d)$	<b>15</b>	<b>15</b>	16	17	16	<b>15</b>	<b>15</b>	78	75	71	69	68	69	<b>67</b>	<b>4</b>	<b>5</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>	<b>4</b>
Weak heuristic	Maze cost ( $\times 10^3$ )							Grid35 cost ( $\times 10$ )							Baldur cost ( $\times 10^2$ )						
$d$	1	5	10	20	40	80	120	1	5	10	20	40	80	120	1	5	10	20	40	80	120
LSS-LRTA*	385	91	63	40	29	20	16	379	132	98	84	75	71	70	188	56	37	25	17	12	10
RTAA*	385	93	65	44	31	20	16	379	156	112	88	77	72	71	188	87	51	32	21	13	10
LRTA* $_{LS}(5, d)$	100	86						134	119						61	52					
LRTA* $_{LS}(10, d)$	50	51	55					99	95	94					37	26	30				
LRTA* $_{LS}(20, d)$	33	37	34	38				85	84	79	81				19	14	15	16			
LRTA* $_{LS}(40, d)$	22	26	26	27	27			79	76	74	72	73			10	8	8	8	9		
LRTA* $_{LS}(80, d)$	16	17	19	20	19	19		74	72	70	68	68	69		6	6	5	5	6	6	
LRTA* $_{LS}(120, d)$	<b>14</b>	<b>14</b>	15	17	16	15	16	72	70	69	<b>66</b>	67	<b>66</b>	68	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>	<b>5</b>

Table 1: Solution cost for LSS-LRTA\*, RTAA\* and LRTA\* $_{LS}(k, d)$  on different benchmarks and heuristics (first trial).

Strong heuristic	Maze time ( $\times 10^{-3}$ ) sg							Grid35 time ( $\times 10^{-4}$ ) sg							Baldur time ( $\times 10^{-3}$ ) sg						
	$d$	1	5	10	20	40	80	120	1	5	10	20	40	80	120	1	5	10	20	40	80
LSS-LRTA*	580	463	247	238	345	624	844	85	81	80	121	229	433	601	74	95	103	92	89	89	95
RTAA*	580	206	144	144	205	360	484	85	48	52	78	144	267	364	74	75	64	57	54	52	56
LRTA* $_{LS}(5, d)$	233	247						40	49						79	97					
LRTA* $_{LS}(10, d)$	129	152	196					35	44	57					62	70	80				
LRTA* $_{LS}(20, d)$	93	110	151	213				<b>34</b>	44	56	84				47	50	55	63			
LRTA* $_{LS}(40, d)$	<b>84</b>	99	135	204	315			38	47	59	85	144			31	35	39	45	56		
LRTA* $_{LS}(80, d)$	94	104	137	201	320	510		42	52	63	90	149	254		<b>28</b>	31	34	39	47	57	
LRTA* $_{LS}(120, d)$	106	115	144	208	334	522	666	46	56	67	93	151	259	336	<b>28</b>	32	35	38	45	55	64
Weak heuristic	Maze time ( $\times 10^{-3}$ ) sg							Grid35 time ( $\times 10^{-4}$ ) sg							Baldur time ( $\times 10^{-3}$ ) sg						
$d$	1	5	10	20	40	80	120	1	5	10	20	40	80	120	1	5	10	20	40	80	120
LSS-LRTA*	776	398	276	272	410	702	977	83	104	118	177	301	557	790	106	97	98	98	97	104	115
RTAA*	776	139	126	141	214	356	510	83	<b>43</b>	52	80	144	270	389	106	76	62	57	58	61	67
LRTA* $_{LS}(5, d)$	187	217						<b>43</b>	54						81	98					
LRTA* $_{LS}(10, d)$	116	157	200					44	58	75					66	69	82				
LRTA* $_{LS}(20, d)$	<b>101</b>	133	169	239				58	71	87	127				49	55	63	69			
LRTA* $_{LS}(40, d)$	<b>101</b>	127	166	253	370			87	95	112	155	229			36	40	44	52	68		
LRTA* $_{LS}(80, d)$	117	132	172	262	423	642		137	144	155	201	290	428		<b>34</b>	36	40	48	60	78	
LRTA* $_{LS}(120, d)$	134	146	181	275	439	691	870	187	192	201	241	341	489	619	40	40	44	51	62	79	92

Table 2: Total planning time for LSS-LRTA\*, RTAA\* and LRTA\* $_{LS}(k, d)$  on different benchmarks and heuristics (first trial).

differ, see column (3). And they also differ in the movement strategy: while LSS-LRTA\* moves to the first state of *OPEN*, always unchanged by the updating strategy of this algorithm, LRTA\*( $k, d$ ) moves to the first state of *OPEN*'.

## 4 Experimental Results

We compare the performance of LRTA\* $_{LS}(k, d)$  with LSS-LRTA\*, RTAA\* and LRTS( $\gamma = 1, T = \infty$ ). Parameter  $d$  is the size of *CLOSED*, and determines the size of the local search space for the three first algorithms,  $d = 1, 5, 10, 40, 80, 120$ . For LRTS, we have used the values 2, 3, 4, 6 for the lookahead depth. Parameter  $k$  is the maximum size of the local learning space for LRTA\* $_{LS}(k, d)$ ,  $k = 5, 10, 20, 40, 80, 120$ . We have used the following benchmarks: (i) Grid35, grids of size  $301 \times 301$  with a 35% of obstacles placed randomly; (ii) Maze, acyclic mazes of size  $181 \times 181$  whose corridor structure was generated with depth-first search; (iii) Baldur's Gate<sup>3</sup>, we have used 5 maps with 2765, 7637, 13765, 14098 and 16142 free states, respectively.

Maze and Grid35 are synthetic 4-connected grids, while Baldur's Gate is an 8-connected grid from a computer game. Actions moving north, east, south, west cost 1, while diago-

nal moves cost  $\sqrt{2}$ . The agent can sense adjacent states to the current one, 4 in Maze and Grid35, 8 in Baldur's Gate (visibility radius = 1). We have used two heuristics, strong and weak, computed as follows. If  $\Delta x = |x_{goal} - x|$ , for Maze and Grid35 the strong heuristic is  $h(x, y) = \Delta x + \Delta y$  (Manhattan distance), and for Baldur's Gate it is the octile distance  $h(x, y) = \sqrt{2} \cdot \min(\Delta x, \Delta y) + |\Delta x - \Delta y|$ . The weak heuristic is  $h(x, y) = \max(\Delta x, \Delta y)$ . Start and goal are chosen randomly assuring that there is a solution path between them. Results consider solution cost and total planning time, averaged over 1,500 instances (10,000 for Baldur's Gate).

First trial results for LSS-LRTA\*, RTAA\* and LRTA\* $_{LS}(k, d)$  appear in Tables 1 (cost) and 2 (time). For a fair comparison, we present results of LRTA\* $_{LS}(k, d)$  with  $k \geq d$  (that is,  $k$ , the upper limit of states to update per planning step, is equal to or higher than  $d$ , the actual limit of the other two algorithms). About LRTS( $\gamma = 1, T = \infty$ ), its results are clearly worse than LRTA\* $_{LS}(k, d)$ , and they are omitted for space reasons. From this point on, we limit the discussion to LSS-LRTA\*, RTAA\* and LRTA\* $_{LS}(k, d)$ .

Regarding solution cost, since LSS-LRTA\* obtains slightly better costs than RTAA\* we limit the discussion to the former. LSS-LRTA\* cost decreases as  $d$  increases, so with more lookahead it obtains better costs. In LRTA\* $_{LS}(k, d)$  parameter  $d$  has much less influence. For any  $d$ , the cost decreases

<sup>3</sup>Baldur's Gate is a registered trademark of BioWare corporation. See [www.bioware.com/games/baldur\\_gate](http://www.bioware.com/games/baldur_gate).

Grid35, Strong heuristic																								
$d$	$h$ increment per update ( $\times 10^{-1}$ )						#updates ( $\times 10^2$ )						#calls shortest paths ( $\times 10$ )						#updates per call					
	1	5	10	20	40	80	1	5	10	20	40	80	1	5	10	20	40	80	1	5	10	20	40	80
LSS-LRTA*	20	18	17	13	8	5	19	21	24	33	55	101	193	47	24	20	14	13	1	5	10	20	40	80
LRTA* <sub>LS</sub> (5, $d$ )	24	24					18	18					48	41					4	4				
LRTA* <sub>LS</sub> (10, $d$ )	26	26	26				18	18	18				29	26	23				6	7	8			
LRTA* <sub>LS</sub> (20, $d$ )	27	27	27	27			19	20	20	20			20	19	17	16			10	11	12	12		
LRTA* <sub>LS</sub> (40, $d$ )	28	28	28	28	27		23	24	25	24	23		18	16	15	14	14		13	15	16	17	17	
LRTA* <sub>LS</sub> (80, $d$ )	28	28	28	29	29	28	27	29	29	29	29	28	16	15	14	13	13	13	17	19	21	22	22	22

Table 3: Measures of LSS-LRTA\* and LRTA\*<sub>LS</sub>( $k, d$ ) execution on Grid35 with strong heuristic (on average, first trial).

with  $k$ , so with more propagation this algorithm obtain better costs. For  $k = d$ , the cost obtained by LSS-LRTA\* is greater than the cost of LRTA\*<sub>LS</sub>( $d, d$ ) (equal for Maze/strong and  $d = 120$ ). When both algorithms allow the same limit of states to update per planning step, LRTA\*<sub>LS</sub>( $d, d$ ) achieves lower costs than LSS-LRTA\* for low and medium  $d$ , and similar costs for high  $d$ . For any  $k \geq d$ , the best cost of LSS-LRTA\* is greater than (but close to) the best cost of LRTA\*<sub>LS</sub>( $k, d$ ) (equal for Maze/strong). In LRTA\*<sub>LS</sub>( $k, d$ ), increasing  $d$  we obtain less improvement than increasing  $k$ , so if we allow  $\delta$  extra states it is more beneficial to add them to the local learning space than to the local search space.

Regarding total planning time, both LSS-LRTA\* and RTAA\* times decrease with  $d$  until some point, from which they increase again; the best time occurs for medium  $d$ . LRTA\*<sub>LS</sub>( $k, d$ ) time increases with  $d$ . In addition, time decreases with  $k$  until some point, from which it increases again. Combining these two tendencies, LRTA\*<sub>LS</sub>( $k, d$ ) best time appears with low  $d$  and medium  $k$ . For  $k = d$ , LSS-LRTA\* time is higher than LRTA\*<sub>LS</sub>( $d, d$ ) time (with the exception of Baldur’s Gate, strong/weak  $d = 5$ ). On the other hand, RTAA\* time is lower than LRTA\*<sub>LS</sub>( $d, d$ ) time (with a few exceptions). For any  $k \geq d$ , RTAA\* best time is higher than LRTA\*<sub>LS</sub>( $k, d$ ) best time, with the exception of Grid35 with weak heuristic, for which both have the same best time. It is worth noting that the solution cost obtained by RTAA\* in its best time  $d$  is clearly worse than the solution cost obtained by LRTA\*<sub>LS</sub>( $k, d$ ) in this best time  $k$  and  $d$ .

In summary, for LSS-LRTA\* to obtain good cost solutions  $d$  is required to be high, but this causes long planning times. LRTA\*<sub>LS</sub>( $k, d$ ) obtains good cost solutions with high  $k$ , requiring much shorter times. On the other hand, RTAA\*

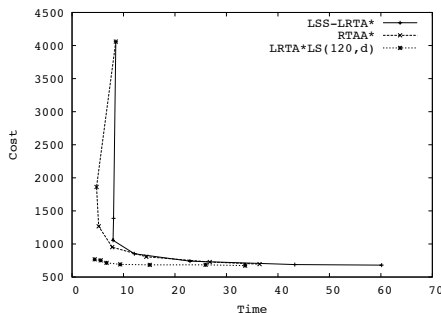


Figure 5: Plot cost-time of LSS-LRTA\*, RTAA\* and LRTA\*<sub>LS</sub>(120,  $d$ ), for  $d = 1, 5, 10, 20, 40, 80, 120$ . We observe that LRTA\*<sub>LS</sub>(120,  $d$ ) dominates other algorithms.

achieves acceptable planning times with medium  $d$  (due to its special update strategy, it does not use shortest paths), but solution costs are clearly worse than those obtained by LRTA\*<sub>LS</sub>( $k, d$ ) with high  $k$ . From these results, it is clear that LRTA\*<sub>LS</sub>( $k, d$ ) offers the best combination solution cost/time, so it is the algorithm of choice. This can be seen in Figure 5, where we plot cost versus time of computing first solution of Grid35 benchmark with strong heuristic for LSS-LRTA\*, RTAA\* and LRTA\*<sub>LS</sub>(120,  $d$ ). Each line has seven points, corresponding to  $d = 1, 5, 10, 20, 40, 80, 120$ . We observe that LRTA\*<sub>LS</sub>(120,  $d$ ) results (points in the cost/time plot) dominate (in the sense of multiobjective optimization) all points of the other two algorithms. Because of that, we claim that LRTA\*<sub>LS</sub>( $k, d$ ) offers the best combination cost/time with high  $k$ .

Since LSS-LRTA\* and LRTA\*<sub>LS</sub>( $k, d$ ) are related algorithms, it is worth asking the reason of this difference in performance. In Table 3 we present different measures (on average) of the execution of both algorithms on Grid35 with strong heuristic (other benchmarks/heuristic produce similar results). The  $h$  increment caused by LRTA\*<sub>LS</sub>( $k, d$ ) each time a state is updated is higher than the increment caused by LSS-LRTA\*. Both algorithms maintain heuristic admissibility, so LRTA\*<sub>LS</sub>( $k, d$ ) computes heuristic values closer to the ideal  $h^*$  than those computed by LSS-LRTA\*, and these updates are done mostly on the same states (approx. 65% of states updated by LSS-LRTA\* are also updated by LRTA\*<sub>LS</sub>( $d, d$ ), for any  $d$ ). Better heuristic values cause to better direct search towards the goal, obtaining solutions with less cost. Only when  $d$  is high LSS-LRTA\* achieves low cost solutions, but this causes a high number of updates and long planning time. About computational effort, both LSS-LRTA\* and LRTA\*<sub>LS</sub>( $k, d$ ) do the same lookahead with A\* and use the same update algorithm (shortest paths) on different local learning spaces. In Table 3, we see #calls to shortest paths and #updates per call. LRTA\*<sub>LS</sub>( $k, d$ ) requires less calls to shortest paths than LSS-LRTA\*, decreasing the difference with  $d$ . While #updates per call is  $d$  for LSS-LRTA\*, it is much lower for LRTA\*<sub>LS</sub>( $k, d$ ), which means less effort for shortest paths. Although LRTA\*<sub>LS</sub>( $k, d$ ) has the extra work to build the set  $I$ , this is compensated by calling shortest paths less times and updating less states. Globally, the presented approach pays off since LRTA\*<sub>LS</sub>( $k, d$ ) runs clearly faster.

About convergence to optimal paths, LRTA\*<sub>LS</sub>( $k, d$ ) results show clear benefits in solution cost and total planning time with respect to the algorithms discussed here (results not shown).

## 5 Conclusions

We have presented  $LRTA^*_{LS}(k, d)$ , a new,  $LRTA^*$ -based algorithm that combines bounded propagation with lookahead. It generates local learning spaces customized to the heuristic inaccuracies detected during lookahead, which could go beyond the local search space. This is a novelty with respect to other RT algorithms, for which the local learning space was always included in the local search space. Experimentally, on three benchmarks with two different heuristics, we have seen that  $LRTA^*_{LS}(k, d)$  computes solutions of lower cost than its competitors for low and medium lookahead. For high lookahead, all considered algorithms achieve solutions with similar costs. However, high lookahead requires long planning times, which are always longer for the other algorithms. Regarding total planning time,  $LRTA^*_{LS}(k, d)$  best time is higher than (equal to for Grid35/weak) the best time of its competitor algorithms, always producing lower cost solutions.

## 6 Acknowledgements

We thank Vadim Bulitko for sharing with us benchmarks of computer games.

## References

- [Bulitko and Lee, 2006] V. Bulitko and G. Lee. Learning in real time search: a unifying framework. *JAIR*, 25:119–157, 2006.
- [Bulitko *et al.*, 2007] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau. Graph abstraction in real-time heuristic search. *JAIR*, 30:51–100, 2007.
- [Hart *et al.*, 1968] P. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 2:100–107, 1968.
- [Hernandez and Meseguer, 2005] C. Hernandez and P. Meseguer.  $Lrta^*(k)$ . In *Proc. of 19th IJCAI*, pages 1238–1243, 2005.
- [Hernandez and Meseguer, 2007] C. Hernandez and P. Meseguer. Improving  $Lrta^*(k)$ . In *Proc. of 20th IJCAI*, pages 2312–2317, 2007.
- [Koenig and Likhachev, 2002] S. Koenig and M. Likhachev.  $D^*$ lite. In *Proc. of AAAI*, pages 476–483, 2002.
- [Koenig and Likhachev, 2006] S. Koenig and M. Likhachev. Real-time adaptive  $a^*$ . In *Proc. of AAMAS*, pages 281–288, 2006.
- [Koenig and Sun, 2009] S. Koenig and X. Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and Multi-Agent Systems*, 2009.
- [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. of AAMAS*, pages 864–871, 2004.
- [Korf, 1990] R. E. Korf. Real-time heuristic search. *Artificial Intelligence*, 42(2-3):189–211, 1990.
- [Stentz, 1995] A. Stentz. The focussed  $d^*$  algorithm for real-time replanning. In *Proc. of IJCAI*, pages 1652–1659, 1995.