Combining Lookahead and Propagation in Real-Time Heuristic Search

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

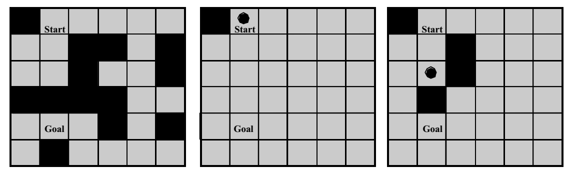
Pedro Meseguer
IIIA, CSIC
Campus UAB, 08193 Bellaterra, Spain
pedro@iiia.csic.es

Abstract

Real-time search methods allow an agent to perform path-finding tasks in unknown environments. Some real-time heuristic search methods may plan several elementary moves per planning step, requiring lookahead greater than inspecting inmediate successors. Recently, the propagation of heuristic changes in the same planning step has been shown beneficial for improving the performance of these methods. In this paper, we present a novel approach that combines lookahead and propagation. Lookahead uses the well-known A* algorithm to develop a *local search space* around the current state. If the heuristic value of a state inspected during lookahead changes, a *local learning space* is constructed around that state in which this change is propagated. The number of actions planned per step depends on the quality of the heuristic found during lookahead: one action if some state changes its heuristic, several actions otherwise. We provide experimental evidence of the benefits of this approach, with respect to other real-time algorithms on existing benchmarks.

Introduction

Let us consider an agent who has to perform a path-finding task from a start position to a goal position in an unknown environment. It can only sense the surrounding area within the range of its sensors (visibility radius), and remembers previously visited positions. An example of this task appears in Figure 1. This situation may happen in real-time computer games (Bulitko & Lee 2006) and control in robotics (Koenig 2001). Off-line search, like A* (Hart, Nilsson, & Raphael 1968) and IDA* (Korf 1985) are not appropriate because they require to know the terrain in advance. Incremental A* methods, like D* (Stentz 1995) and D*Lite (Koenig & Likhachev 2002), and real-time search methods (Korf 1990) are adequate. (a comparison between incremental versions of A* and real-time heuristic search appears in (Koenig 2004)). If the first-move delay (the time required for the agent to start moving) is required to be short, incremental A* methods are discarded because they require to compute the complete solution before start moving, which may be long. Real-time heuristic search remains the only applicable strategy for this task.



(i) (ii) (iii)

Figure 1: An example of path-finding task: (i) the exact map (ii) what the agent, represented by a dot, knows at the beginning (iii) after some steps, the agents knows more about the exact map, which is partially revealed.

Real-time 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 (in this paper we assume that an action produces an elementary move from an state to one of its successors). The agent has a short time to perform the planning phase. Due to this hard requirement, real-time methods restrict search to a small part of the state space around the current state, called the *local search space*. The size of this space is small and independent of the size of the complete state space. Searching in this local space is usually called *lookahead*, that is feasible in the limited planning time. The agent finds how to move in this local space and plans one or several actions, to be performed in the next action execution phase. It is debatable if planning one action is better than planning several actions per planning step, with the same lookahead. The whole process iterates with new planning and action execution phases until a goal is found.

At each step, real-time methods compute the beginning of the trajectory from the current state to a goal. Search is limited to a small portion of the state space, so there is no guarantee to produce an optimal trajectory. Some methods guarantee that after repeated executions on the same instance, the trajectory converges to an optimal path. Real-time methods update heuristic values of some states. Propagation of these changes has improved the performance of these methods.

In this paper, we present a novel approach to real-time search that combines lookahead, bounded propagation and dynamic selection of one or several actions per planning step. Previously, bounded propagation considered heuristic changes at the current state only. Now, if lookahead is allowed, we extend the detection and repair (including bounded propagation) of heuristic changes to any expanded state. In addition, we consider the quality of the heuristic found during lookahead as a criterion to decide between one or several actions to perform per planning step. We implement these ideas in the LRTA*$_{LS}(k, d)$ algorithm, which empirically shows a better performance than other real-time search algorithms on existing benchmarks.

The structure of the paper is as follows. We define the problem and summarize some solving approaches. We present our approach and the LRTA*$_{LS}(k, d)$ algorithm, with empirical results. Finally, we extract some conclusions from this work.

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

A real-time search algorithm governs the behavior of an agent, which contains in its memory a map of the enviroment (memory map). Initially, the agent does not know where the obstacles are, and its memory map contains the initial heuristic. The agent is located in a specific (current) state and it senses the environment, identifying obstacles within its sensors range (visibility radius). This information is uploaded in its memory map, performing the *planning phase*:

- Lookahead: the agent performs a local search around the current state in its memory map.
- Learning: if better (more accurate) heuristic values are found during lookahead, these values are backed-up, causing to change the heuristic of one or several states.
- Action selection: according to the memory map, one or several actions are selected for execution.

In practice, only states which change its heuristic value are kept in memory (usually in a hash table), while the value of the other states can be obtained using the initial heuristic function. 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. Notice that obstacles are discovered by the agent (and registered in its memory map) as the agent goes close to them and they are detected by its sensors. Following (Bulitko *et al.* 2007), most existing real-time search algorithms can be described in terms of:

- *Local search space*: those states around the current state 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 the heuristic value of a state in the local learning space changes.
- *Control strategy*: how to select the next action to perform.

Some well-known real-time search algorithms are summarized in Figure 2 (for details the reader should consult the original sources). For simplicity, we assume that the distance between any two neighbor states is 1. LRTA* (Korf 1990), possibly the most popular real-time search algorithm, is considered in its simplest form 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 $\{t|dist(x, t) \leq d\}$ and the learning rule is the minimin strategy, while the local learning space and control selection remain unchanged. In a state space like the one assumed here (finite, minimum positive costs, finite heuristic 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 values computed in the previous trial), the heuristic converges to its exact values along every optimal path (random tie-breaking) (Korf 1990).

Regarding LRTA*(k) (Hernandez & 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 sucessors, etc.). To limit computation time, propagation is bounded: up to a maximum of k states can be reconsidered per planning step. This is implemented using the queue Q, where a maximum of k states could enter. The idea of bounded propagation is better developed in LRTA*$_{LS}(k)$ (Hernandez & Meseguer 2007). If the current state changes, the local learning space I is constructed from all states which may change its heuristic as consequence of that change. The learning rule uses the shortest path algorithm from F, the frontier of I.

Koenig proposed a version of LRTA* (Koenig 2004) which performs lookahead using A*, until the number of states in *CLOSED* reaches some limit. At this point, the heuristic of every state in *CLOSED* is updated following a shortest paths (Dijkstra) algorithm from the states in *OPEN*. Looking for a faster updating, a new version called RTAA*

	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*$_{LS}(k)$	$\{x\} \cup succ(x)$	I	$max\{h(y), min_{f \in F}[k(y,f) + h(f)]\}$ where $k[y,f]$ is the shortest distance from y to f in I	move to best successor
LRTA* (Koenig)	*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 shortest distance from y to f in *CLOSED*	move to best state in *OPEN* by the shortest 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 shortest path in *CLOSED*
LRTS	$\{t \mid dist(x,t) \leq d\}$	$\{x\}$	$max\{h(x), [k(x,w) + h(w)]\}$, where w is $argmax_{i=1,\ldots,d} min_{t \mid dist(x,t)=i}[k(x,t) + h(t)]$	move to best state at distance d by the shortest path in LSS

Figure 2: Summary of some real-time search algorithms: x is the current state, y is any state of the local learning space.

was proposed (Koenig & Likhachev 2006). 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 selected as the next state to move, which may involve several elementary moves.

Bulitko and Lee proposed the LRTS algorithm (Bulitko & 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 maximum of the minima at each level is taken for update). The control strategy selects the best state at level d as the next state to move, which may involve several elementary moves.

Ii is assumed that moves are computed with the *free space assumption*: if a state is not within the agent sensor range (visibility radius) and there is no memory of an obstacle, that state is assumed feasible. When moving, if an obstacle is found in a feasible state, execution stops and another planning starts.

Moves and Lookahead: LRTA*$_{LS}(k, d)$

Lookahead greater than visiting inmediate successors is required for planning several actions per step. In this case, more opportunities exist for propagation of heuristic changes. Previous approaches including bounded propagation consider changes in the heuristic of the current state only. Now, the detection and propagation of heuristic changes can be extended to any state expanded during lookahead. In this way, heuristic inaccuracies found in the local search space can be repaired, no matter where they are located inside that space. To repair a heuristic inaccuracy, we generate a local learning space around it and update the heuristic of its states, as done in the procedure LS presented in (Hernandez & Meseguer 2007). Independently of bounded propagation, two types of lookahead are considered: *static lookahead*, when the whole local search space is traversed independently of the heuristic, and *dynamic lookahead*, when traversing the local search space stops as soon as there is evidence that the heuristic of a state may change.

There is some debate around the adequacy of planning one action versus several actions per planning step, with the same lookahead. Typically, single-action planning produces trajectories of minor cost. However, the overall CPU time devoted to planning in single-action planning is usually longer than in several actions planning, since the whole lookahead effort produces a single move. Nevertheless, planning several actions is an attractive option that has been investigated in different settings (Koenig 2004), (Bulitko & Lee 2006).

Planning a *single* action per step is conservative. The agent has found the best trajectory in the local search space. But from a global perspective, it is unsure whether this trajectory effectively brings the agent closer to a goal or to an obstacle (if the path is finally wrong this will become apparent after some moves, when more parts of the map are revealed). In this situation, the least commitment is to plan a single action: the best move from the current state.

Planning *several* actions per step is risky, for similar reasons. Since the local search space is a small fraction of the whole search space, it is unclear if the best trajectory at local level is also good at global level. If it is finally wrong, some effort is required to come back (several actions to undo). But if the trajectory is good, performing several actions in one step will bring the agent closer to the goal than performing a single move.

These are two extremes of a continuum of possible planning strategies. In addition, we propose the *dynamic* approach, that consist in taking into account the quality of the heuristic found during lookahead. If there is some evidence that the heuristic quality is not perfect at local level, we do not trust the heuristic values and plan one action only. Otherwise, if the heuristic quality is perfect at local level, we trust it and plan several actions. Specifically, we propose not to trust the heuristic when one of the following conditions holds:

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

These ideas are implemented in the LRTA*$_{LS}(k, d)$ algorithm, that includes the following features:

- Local search space. Following (Koenig 2004), lookahead is done using A* (Hart, Nilsson, & Raphael 1968). It stops when (i) $|CLOSED| = d$ (static lookahead) or (ii) when an heuristic inaccuracy is detected, although $|CLOSED| < d$ (dynamic lookahead), or (iii) when a gal is found. The local search space is formed by *CLOSED* $\cup$ *OPEN*.

- Local learning space. When the heuristic of a state x in *CLOSED* $\cup\{$best-state(*OPEN*)$\}$ changes, the local space (I, F) around x (I is the set of interior states and F is the set of frontier states) is computed using the procedure presented in (Hernandez & Meseguer 2007). $|I| \leq k$.
- Learning rule. Once the local learning space I is selected, propagation of heuristic changes in I is done using the Dijkstra shortest paths algorithm, as done in (Koenig 2004).
- Control strategy. We consider three possible control strategies: single action, several actions or dynamic.

Since there are two options for lookahead and three possible control strategies, there are six different versions of LRTA*$_{LS}(k, d)$. Their relative performance are discussed in the next section. Before considering specific details, we want to stress the fact that, in LRTA*$_{LS}(k, d)$, the local search and local learning spaces are developed independently (parameters d and k determine the sizes of both spaces). There is some intersection between them, but the local learning space is not necessarily included in the local search space (in fact, this is true for all algorithms that include bounded propagation, check the local search and local learning spaces in Figure 2). This is a novelty, because the local learning space was totally included in the local search space of other real-time search algorithms.

LRTA*$_{LS}(k, d)$ with dynamic lookahead and dynamic control 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 2). This procedure works at follows. First, it performs lookahead from the current state x using the A* algorithm (line 3). A* performs lookahead until (i) it finds a state which heuristic value satisfies the updating condition, (ii) there are d states in *CLOSED*, or (iii) it finds a solution state. In any case, it returns the sequence of states, $path$, that starting with the current state x connects with (i) the state which heuristic value satisfies the updating condition, (ii) the best state in *OPEN*, or (iii) a solution state. Observe that $path$ has at least one state x, and the only state that might change its heuristic value is $\texttt{last}(path)$. If this state satisfies the updating condition (line 4), then this change is propagated: the local learning space is determined and updated using the Dijkstra shortest paths algorithm (line 5). One action is planned and executed (lines 6-7), and the loop iterates. If $\texttt{last}(path)$ does not change its heuristic, actions passing from one state to the next in $path$ are planned and executed (lines 9-12).

Function $\texttt{SelectLS}(k, d)$ computes (I, F). I is the local learning space around x, and F surrounds I immediate and completely. This function keeps queue Q that contains state candidates to be included in I or F. Q is initialized with the current state x and I and F are empty (line 1). At most k states will enter I, controlled by the counter $cont$. A loop is executed until Q contains no elements or $cont$ is equal to k (**while** loop, line 2). The first state v of Q is extracted (line 3). The state $y \leftarrow \text{argmin}_{w \in Succ(v) \wedge w \notin I}[c(v, w) + h(w)]$ is computed (line 4). If v is going to change (line 5), it enters I and the counter increments (line 6). Those successors of v which are not in I or Q enter Q by rear (line 8). If v does not change, v enters F (line 9). When exiting the loop, if Q still contains states, they are added to F. Function $\texttt{Changes?}(x)$ returns *true* if x satisfies the updating condition, *false* otherwise (line 2).

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 　$path \leftarrow \texttt{A*}(x, d, G)$; $z \leftarrow \texttt{last}(path)$;
4 　**if** $\texttt{Changes?}(z)$ **then**
5 　　$(I, F) \leftarrow \texttt{SelectLS}(z, k)$; $\texttt{Dijkstra}(I, F)$;
6 　　$y \leftarrow \text{argmin}_{v \in Succ(x)}[c(x, v) + h(v)]$;
7 　　$\texttt{execute}(a \in A$ such that $a = (x, y))$; $x \leftarrow y$;
8 　**else**
9 　　$x \leftarrow \texttt{extract-first}(path)$;
10 　　**while** $path \neq \emptyset$ **do**
11 　　　$y \leftarrow \texttt{extract-first}(path)$;
12 　　　$\texttt{execute}(a \in A$ such that $a = (x, y))$; $x \leftarrow y$;

function $\texttt{SelectLS}(x, k)$: `pair of sets`;
1 $Q \leftarrow \langle x \rangle$; $F \leftarrow \emptyset$; $I \leftarrow \emptyset$; $cont \leftarrow 0$;
2 **while** $Q \neq \emptyset \wedge cont < k$ **do**
3 　$v \leftarrow \texttt{extract-first}(Q)$;
4 　$y \leftarrow \text{argmin}_{w \in Succ(v) \wedge w \notin I}[c(v, w) + h(w)]$;
5 　**if** $h(v) < c(v, y) + h(y)$ **then**
6 　　$I \leftarrow I \cup \{v\}$; $cont \leftarrow cont + 1$;
7 　　**for each** $w \in Succ(v)$ **do**
8 　　　**if** $w \notin I \wedge w \notin Q$ **then** $Q \leftarrow \texttt{add-last}(Q, w)$;
9 　**else if** $I \neq \emptyset$ **then** $F \leftarrow F \cup \{v\}$;
10 **if** $Q \neq \emptyset$ **then** $F \leftarrow F \cup Q$;
11 **return** (I, F);

function $\texttt{Changes?}(x)$: `boolean`;
1 $y \leftarrow \text{argmin}_{v \in Succ(x)}[c(x, v) + h(v)]$;
2 **if** $h(x) < c(x, y) + h(y)$ **then return** $true$; **else return** $false$;

Figure 3: The LRTA*$_{LS}(k, d)$ algorithm, with dynamic lookahead and dynamic control.

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 in the same terms as LRTA* (Theorem 3 (Korf 1990)). LRTA*$_{LS}(k, d)$ inherits the good properties of LRTA*.

One might expect that LRTA*$_{LS}(k, d)$ collapses into LRTA*$_{LS}(k)$ when $d = 1$. It is almost the case. While in LRTA*$_{LS}(k)$ the only state that may change is x (the current state), in LRTA*$_{LS}(k, d = 1)$ it may also change the best of successors of x (best state in *OPEN*).

Experimental Results

We have six versions of LRTA*$_{LS}(k, d)$ (from two lookahead options times three control strategies). Experimentally we have seen that the version selecting always a single action produces the lowest cost, both in first trial and convergence, while the version selecting always several actions requires the lowest time, also in first trial and convergence. A reasonable trade-off between cost and time occurs for the version with dynamic lookahead and dynamic selection of the number of actions depending on the heuristic quality. In the following, we present results of this version.

We compare the performance of LRTA*$_{LS}(k,d)$ with LRTA* (version of Koenig), RTAA* and LRTS($\gamma = 1$,T$= \infty$). Parameter d is the size of the *CLOSED* list in A*, and determines the size of the local search space for the three first algorithms. We have used the values $d = 2,4,8,16,32,64,128$. 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*$_{LS}(k,d)$. We have used the values $k = 10,40,160$.

To evaluate algorithmic performance, we have used synthetic and computer games benchmarks. Synthetic benchmarks are four-connected grids, on which we use Manhattan distance as the initial heuristic. We have used the following ones:

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

Computer games benchmarks are built from different maps of two commercial computer games:

1. Baldur's Gate [1]. We have used 5 maps with 2765, 7637, 13765, 14098 and 16142 free states, respectively.
2. WarCraft III [2]. We have used 3 maps, which have 10242, 10691 and 19253 free states, respectively.

In both cases, they are 8-connected grids and the initial heuristic of cell (x,y) is $h(x,y) = max(|x_{goal} - x|, |y_{goal} - y|)$. In synthetic and computer games benchmarks, the start and goal states are chosen randomly assuring that there is a path from the start to the goal. All actions have cost 1. The visibility radius of the agent is 1 (that is, agent sensors can "see" what occur in inmediate neighbors of the current state). Results consider first trial and convergence to optimal trajectories in solution cost (= number of actions performed to reach the goal) and total planning time (in milliseconds), plotted against d, averaged over 1500 instances (Grid35 and Maze), 10000 instances (Baldur's Gate) and 6000 instances (WarCraft III). We present results for Maze and Baldur's Gate only (the other two benchmarks show similar results).

LRTA*, RTAA* and LRTA*$_{LS}(k,d)$ results appear in Figure 4, while LRTS($\gamma = 1$,T$= \infty$) results appear in Table 1. They are not included in Figure 4 for clarity purposes. Solution costs and planning times are worse than those obtained by LRTA*$_{LS}(k,d)$. From this point on, we limit the discussion to LRTA*, RTAA* and LRTA*$_{LS}(k,d)$.

[1]Baldur's Gate is a registered trademark of BioWare corporation. See www.bioware.com/games/baldur_gate.

[2]WarCraft III is a registered trademark of Blizzard Entertainment. See www.blizzard.com/war3.

Results of first trial on Maze appear in the first and second plots of Figure 4. We observe that solution cost decreases monotonically as d increases, and for LRTA*$_{LS}(k,d)$ it also decreases monotonically as k increases. The decrement in solution cost of LRTA*$_{LS}(k,d)$ with d for medium/high k is very small: curves are almost flat. The best results with low lookahead are for LRTA*$_{LS}(k,d)$, and all algorithms have a similar cost with high lookahead. Regarding total planning time, LRTA*$_{LS}(k,d)$ increases monotonically with d, while LRTA* and RTAA* first decrease and later increase. The best results are for LRTA*$_{LS}(k,d)$ with medium/high k and low d, and for RTAA* with medium d. However, the solution cost of LRTA*$_{LS}(k,d)$ with medium k and low d is much lower than the solution cost of RTAA* with medium d. So LRTA*$_{LS}(k,d)$ with medium k and low d offers the lowest solution cost with the lowest total time.

Results of convergence on Maze appear in the third and forth plots of Figure 4. Regarding solution cost, curve shapes and their relative position are similar to those of first trial on Maze. Regarding total planning time, LRTA*$_{LS}(k,d)$ versions increase monotonically with d (a small decrement with small d is also observed), while LRTA* and RTAA* decrease monotonically with d. The best results appear for LRTA*$_{LS}(k,d)$ with low d and medium/high k, and for LRTA* and RTAA* with high d. Comparing the solution cost of these points, the best results are for LRTA*$_{LS}(k,d)$ with low d and high k, while RTAA* and LRTA* with high d go next. LRTA*$_{LS}(k,d)$ obtains the minimum cost and the shortest time with low d and high k.

Results of first trial on Baldur's Gate appear in the fifth and sixth plots of Figure 4. Regarding solution cost, curve shapes and their relative positions are similar to those of first trial on Maze. Here, we observe that from medium to high lookahead the solution cost of LRTA* and RTAA* are better than the cost obtained by LRTA*$_{LS}(k,d)$ with low k. LRTA*$_{LS}(k,d)$ curves are really flat after the first values of d. Algorithms offering best cost solutions are LRTA*$_{LS}(k,d)$ with medium/high k and any d, followed by LRTA* and RTAA* with high d. Regarding total planning time, curve shapes are also similar to Maze first trial. Here, RTAA* is not so competitive, its best results are not so close to the best results of LRTA*$_{LS}(k,d)$ as in Maze. The best time is for LRTA*$_{LS}(k,d)$ with medium/high k and low d, which also offers a very low solution cost.

Results of convergence on Baldur's Gate appear in the last two plots of Figure 4. Regarding solution cost, curve shapes and their relative positions are similar to those of first trial on the same benchmark. Regarding total plan-

	First trial Maze		Convergence Maze	
	Cost	Time	Cost	Time
2	286,298.0	384.8	9,342,177.6	12,495.8
3	318,744.6	411.0	13,484,604.6	17,103.7
4	346,511.8	439.7	6,213,024.9	7,692,1
6	349,338.4	445.0	6,925,467.7	8,431.9

Table 1: LRTS($\gamma = 1$, T$= \infty$) results. The lookahead depth appears in the leftmost column.

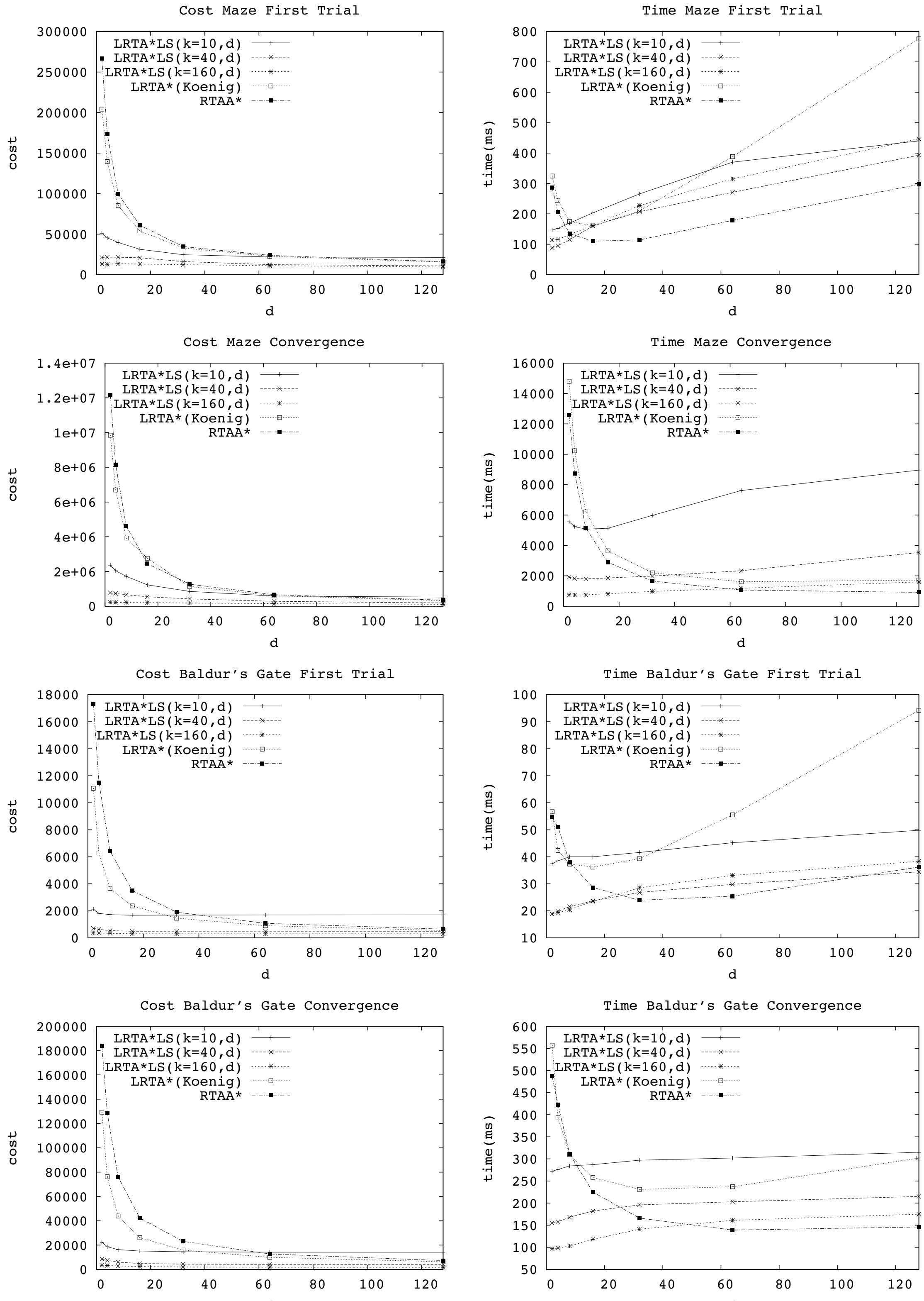


Figure 4: Experimental results on Maze and Baldur's Gate: solution cost and total planning time for first trial and convergence.

ning time, curve shapes are also similar to those of Baldur's Gate first trial. Here, the best results in time are obtained by LRTA*$_{LS}(k,d)$ with high k and low d, followed by RTAA* with high d. The best results in time are also the best results in solution cost, so the clear winner is LRTA*$_{LS}(k,d)$ with high k and low d.

In summary, regarding solution cost we observe a common pattern in both benchmarks, in first trial and convergence. LRTA* and RTAA* solution cost decrease monotonically with d, from a high cost with low d to a low cost with high d. In contrast, LRTA*$_{LS}(k,d)$ offers always a low solution cost for any d (variations with d are really small), and cost decreases as k increases. The best results are obtained for medium/high k and any d. So, if minimizing solution cost is a major concern, LRTA*$_{LS}(k,d)$ is the algorithm of choice. Regarding total planning time, there is also a common pattern in both benchmarks. We observe that LRTA* is always more expensive in time than RTAA*, so we limit the discussion to the latter. LRTA*$_{LS}(k,d)$ with high k (in some cases also with medium k) and low d obtain the shortest times. In those points where RTAA* obtains costs comparable with LRTA*$_{LS}(k,d)$ (points with high d), RTAA* requires much more time than LRTA*$_{LS}(k,d)$ with high k and low d. The only exception is convergence on Maze, where RTAA* time is higher than but close to LRTA*$_{LS}(k,d)$ time. From this analysis, we conclude that LRTA*$_{LS}(k,d)$ with high k and low d is the best performant algorithm, both in solution cost and time.

Let us consider LRTA*$_{LS}(k,d)$ parameters. Regarding solution cost, d does not cause significant changes, while k is crucial to obtain a good solution cost. Regarding total planning time, both d and k are important (d seems to have more impact). The most performant combination is high k and low d. This generates the following conclusion: in these algorithms, it is more productive to invest in propagation than in lookahead. If more computational resources are available in a limited real-time scenario, it seems more advisable to use them for bounded propagation (increase k) than to make larger lookahead (increase d).

Conclusions

We have presented LRTA*$_{LS}(k,d)$, a new real-time algorithm able to plan several moves per planning step. It combines features already introduced in real-time search (lookahead using A*, bounded propagation, use of Dijkstra shortest paths). In addition, it considers the quality of the heuristic to select one or several actions per step. LRTA*$_{LS}(k,d)$ is complete and converges to optimal paths after repeated executions on the same instance. Experimentally, we have seen on several synthetic and computer games benchmarks that LRTA*$_{LS}(k,d)$ outperforms LRTA* (version of Koenig), RTAA* and LRTS($\gamma = 1$,T$= \infty$).

LRTA*$_{LS}(k,d)$ presents several differences with other competitor algorithms. It is good to know which are details and which are matter of substance. LRTA*$_{LS}(k,d)$ belongs to the LRTA* family and is not very far from LRTA* (version of Koenig): (i) both have the same local search space, (ii) both use the same learning rule and (iii) although different, their control strategies are related. Their main difference is the local learning space, *CLOSED* in LRTA* and I in LRTA*$_{LS}(k,d)$. *CLOSED* is a kind of generic local learning space, built during lookahead but not customized to the heuristic depression that generates the updating. However, I is a local learning space that is built specifically to repair the detected heuristic inaccuracy. I may not be totally included in the local search space, something new with respect to the other considered algorithms. According to the experimental results, using customized local learning spaces seems to be more productive than using generic ones. And the size of these customized local learning spaces (upper bounded by k) is significant for the performance of real-time heurisitic search algorithms.

Acknowledgements

We thank Vadim Bulitko for sharing with us benchmarks of computer games, and Ismel Brito for his help with the plots.

References

Bulitko, V., and Lee, G. 2006. Learning in real time search: a unifying framework. *Journal of Artificial Intelligence Research* 25:119–157.

Bulitko, V.; Sturtevant, N.; Lu, J.; and Yau, T. 2007. Graph abstraction in real-time heuristic search. *Journal of Artificial Intelligence Research* 30:51–100.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics* 2:100–107.

Hernandez, C., and Meseguer, P. 2005. Lrta*(k). In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, 1238–1243.

Hernandez, C., and Meseguer, P. 2007. Improving lrta*(k). In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2312–2317.

Koenig, S., and Likhachev, M. 2002. D*lite. In *Proceedings of the National Conference on Artificial Intelligence*, 476–483.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive a*. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 281–288.

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

Koenig, S. 2004. A comparison of fast search methods for real-time situated agents. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems*, 864–871.

Korf, R. E. 1985. Depth-first iterative deepening:an optimal admissible tree search. *Artificial Intelligence* 27:97–109.

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

Stentz, A. 1995. The focussed d* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1652–1659.