Improving Real-Time Heuristic Search on Initially Unknown Maps

Carlos Hernández[1] and Pedro Meseguer[2]

[1]Universidad Católica de la Ssma. Concepción.
Caupolicán 491, Concepción, Chile
[2]Institut d'Investigació en Intel.ligència Artificial, CSIC.
Campus UAB, 08193 Bellaterra, Spain.
{chernan|pedro}@iiia.csic.es

Abstract

Real-time search methods allow an agent to perform path-finding tasks in unknown environments. Some of these methods may plan several actions per planning step. We present a novel approach, where the number of planned actions per step depends on the quality of the heuristic values found in the lookahead. If, after inspecting the neighborhood of a state, its heuristic value changes, only one action is planned. When the heuristic values of all states in the lookahead do not change, several actions are planned. 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 environment that is initially unknown. The agent can only sense the surrounding area that is in range of its sensors. In addition, it remembers those areas that it has visited previously. An example of this task appears in Figure 1. This situation may happen in characters of real-time computer games (Bulitko & Lee 2006) and control in robotics (Koenig 2001). Off-line search methods, like A* (Hart, Nilsson, & Raphael 1968) are not appropriate for these tasks, because they require knowing the terrain in advance. Incremental versions of A*, like D*Lite (Koenig & Likhachev 2002), and real-time search methods (Korf 1990) allow an agent to solve this task, but of these two, only real-time search methods can perform the planning phase in a limited, short amount of time (a comparison between incremental versions of A* and real-time heuristic search appears in (Koenig 2004)).

Real-time search interleaves planning and action execution phases in an on-line manner. In the planning phase the agent plans one or several actions, which are performed in the action execution phase. Real-time methods restrict the search to a small part of the state space that is around the current state. This part is called the *local space*. The size of the local space is small and independent of the size of the complete state space, so that searching in the local space is feasible in the limited time of the planning phase. As a result, the agent determines how to move inside the local space and plans one or several actions, which are performed in the next action execution phase. The whole process iterates with new planning and action execution phases until a goal state is reached.

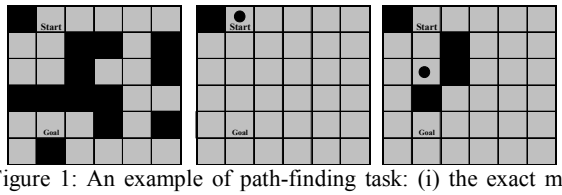

Figure 1: An example of path-finding task: (i) the exact map (ii) what the agent, represented by a dot, knows at the beginning (iii) after some steps, the agents knows more about the exact map, which is partially revealed.

At each step, this strategy computes the beginning of the trajectory from the current state to a goal state. Since search is limited to a small portion of the state space, there is no guarantee to produce an optimal global trajectory. However, some methods guarantee that after repeated executions on the same problem instance, the trajectory converges to an optimal path. To prevent cycling, real-time methods update the heuristic values associated with visited states.

Initially, real-time search algorithms planned one action per planning step (Korf 1990). Nowadays, several actions can be planned at each planning step, actions that are sequentially performed after it (Koenig 2004), (Koenig & Likhachev 2006), (Bulitko & Lee 2006).

There is some debate about the relative performance of planning one single action versus planning several actions per planning step, with the same amount of lookahead. If a single action is planned, the resulting action will probably be of greater quality than if several actions were planned. This may decrease the cost of the final solution. However, planning one action per step often increases the overall CPU time devoted to planning. In this paper we present an alternative approach: to take into account the quality of the heuristic found during lookahead. If we find any evidence that the heuristic is not accurate, we suggest taking a conservative approach planning one action only.

Otherwise, we propose to take a more risky approach, planning of several actions in this step. In addition, if any inaccuracy is detected during lookahead, it is repaired even though the inaccuracy is not located at the current state.

The structure of the paper is as follows. First, we define precisely the problem and summarize some of the most popular approaches to solve it. Then, we present our approach, showing how the quality of the heuristic has to be taken into account when performing lookahead. We implement these ideas in the LRTA*(k, d) algorithm, and we present experimental results comparing this approach and existing algorithms. Finally, we present conclusions from this work, and address issues for further research.

Problem Definition

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 $A \subset X \times X \setminus \{(x, x)\}$, 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 state w. The state space is undirected: for any action $(x, y) \in A$ there exists its inverse $(y, x) \in A$ with the same cost $c(x, y) = c(y, x)$. The 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 $h : X \rightarrow [0, \infty)$ 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 $w \in Succ(x)$. A path $\{x_0, x_1,.., x_n\}$ with $h(x_i) = h^*(x_i)$, $0 \leq i \leq n$ is optimal.

LRTA* is a real-time search algorithm that performs a loop of lookahead, update and action execution, until a goal node is found. If x is the current state, it performs lookahead at depth 1, computing $y = \arg\min_{z \in Succ(x)} c(x, z) + h(z)$ (lookahead at greater depths can also be computed, this is the simplest version). If $h(x) < c(x, y) + h(y)$ (we call it the *updating condition*), LRTA* updates $h(x)$ to $c(x, y) + h(y)$ and moves to the state y. In a state space like the one assumed here (finite, minimum positive costs, finite heuristic estimates) where from every state there is a path to a goal, it has been proved that LRTA* is complete. In addition, if h is admissible, over repeated trials (where each trial takes as input the heuristic values computed in the previous trial), the heuristic estimates eventually converge to their exact values along every optimal path (with random tie-breaking) (Korf 1990).

Related Work

In RTA* / LRTA* (Korf 1990), the merit of a state x is $f(x) = g(x) + h(x)$, where $g(x)$ is the distance from the current state to x (in contrast with A*, where $g(x)$ is the distance from the initial node to x). Both algorithms were initially presented performing one move per step, but it can be easily modified to compute d moves per step, by expanding d levels of the search tree rooted at the current state. RTA* / LRTA* updates the heuristic of the current state with the 2nd min / 1st min of f at depth d. After updating, it executes d moves to the best state of the frontier (state with minimum f at depth d).

The LRTA* version of (Koenig 2004) searches the local space around the current state using the A* algorithm. It uses a parameter k that limits the number of states that enter the CLOSED list. When this limit is achieved, all nodes in CLOSED are updated using a shortest path algorithm. After updating, the agent performs several moves to reach the first node in the OPEN list. The RTAA* algorithm (Koenig & Likhachev 2006) follows a similar strategy with a different updating mechanism, which is simpler but less informed than the one used in the LRTA* version of (Koenig 2004).

The LRTS algorithm (Bulitko & Lee 2006) also computes d moves. As in the previous case, it expands d levels of the search tree, but the updating process is different. LRTS updates the current state with the maximum value among the states with minimum f at each depth between 1 and d. After updating, the agent performs d moves to the state with minimum f at depth d.

The LRTA*$_{LS}(k)$ of (Hernandez & Meseguer 2007) plans a single action per step. If the heuristic of the current state changes, it propagates this change in the local space computed by a special procedure and composed up to k states.

In unknown environments, moves are computed using the "free space assumption": if a state is not in the visibility range of the agent and there is no evidence that it contains an obstacle, it is assumed to be feasible. When moves are performed, if an obstacle is found in one of these assumed feasible states, the execution stops and a new planning phase starts.

Several Moves per Step

As mentioned above, there is some debate around the adequacy of planning one action versus planning several actions per planning step, with the same amount of lookahead. Typically, single-action planning produces trajectories of better quality (that is, minor cost). However, the overall CPU time devoted to planning in single-action planning is usually longer than in the other approach, since the whole effort of lookahead produces a single move. Thus, 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 a conservative strategy. The agent has searched the local space and has found the best trajectory in it. But, from a global perspective, the agent is unsure whether this best trajectory effectively brings the agent closer to a goal state, or if it follows a wrong path moving closer to an obstacle barrier (if the path is finally wrong this will become apparent after

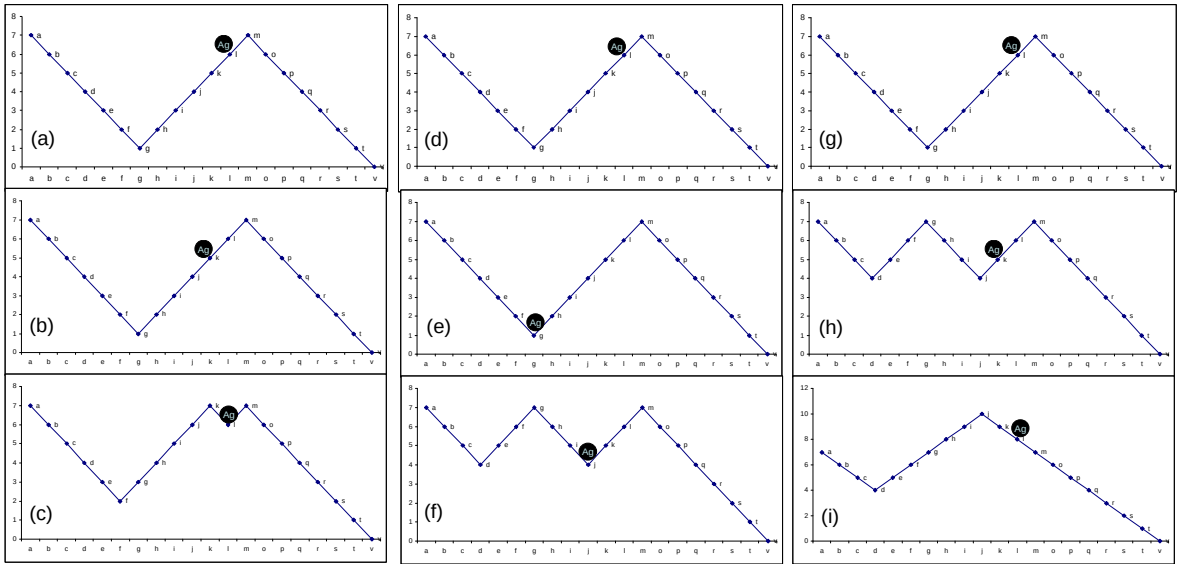

Figure 2: Example of moving strategies in real-time search for an agent, represented as a black dot, in unidimensional space. States are represented in the horizontal axis, while heuristic values appear in the vertical axis. The solution state is *v*. local space has 5 states, up to 5 actions can be planned per step. Left: single-move strategy. Center: several moves strategy. Right: the strategy proposed in this paper.

some moves, when more parts of the map are revealed). In this situation, the least commitment strategy is to plan a single action: the best move from the current state.

Planning several actions per step can be seen as a risky strategy, for the same reasons. Since the local space is small with respect to the whole state space, it is unclear if the best trajectory inside the local space is also good at a global level. Following the best trajectory in the local space is risky because (i) it might not be good at a global level, and (ii) if it is finally wrong, since it includes several actions, it will require some effort to come back. Otherwise, if the trajectory is good, performing several actions in one step will bring the agent closer to the goal than will performing a single move.

These strategies are two extremes of a continuum of possible planning strategies. We propose an intermediate position between them, which consists on taking into account the quality of the heuristic found during lookahead. If there is any evidence that the heuristic quality is not perfect at local level, we do not trust the heuristic values: we take a conservative approach and plan one action only. Otherwise, if the heuristic quality is perfect in the local space, we trust it, taking the more risky approach and planning for several actions.

Specifically, we propose not to trust the heuristic when one of the following conditions holds:

1. the final state for the agent (= the best state in the frontier of the local space) satisfies the updating condition,
2. there is a state in the local space that satisfies the updating condition.

In both cases, we repair the inaccuracy of the heuristic, that is, we generate a local space around that state, we update the heuristic and propagate this change following the bounded propagation method (Hernandez & Meseguer 2007). This is an important point in our approach: as soon as one heuristic inaccuracy is detected, it is repaired and propagated.

An example of the proposed strategy appears in Figure 2. We present the first two planning and execution steps of an agent in one-dimensional space, where the cost of moving to a neighbor state is 1, states appear on the horizontal axis and heuristic values are shown on the vertical axis. This agent performs lookahead using A* until CLOSED has 5 states, it updates heuristic values of states in CLOSED using a shortest path algorithm and it is able to plan up to 5 actions per step. The single-action strategy is shown on the left (graphics a, b and c). Initially in (a), the agent is in state *l*. It performs A* until CLOSED = {*h*, *i*, *j*, *k*, *l*}. The shortest path algorithm on CLOSED causes no change, and the best action is $l \rightarrow k$ (b). Then, the process repeats: A* executes until CLOSED = {*g*, *h*, *i*, *j*, *k*}, the shortest path algorithm causes the changes in heuristic values that appear in (c) and the agent plans the best action $k \rightarrow l$ (depending on tiebreaking, it could also be $k \rightarrow j$). In the center graphs (d, e and f), the planning several actions per step strategy is shown. Initially in (d), the agent is in state *l*. It performs A* until CLOSED = {*h*, *i*, *j*, *k*, *l*}. The shortest path algorithm on CLOSED causes no change. Then, it plans five actions: $l \rightarrow k \rightarrow j \rightarrow i \rightarrow h \rightarrow g$. So the agent moves finally to *g* (e) and the whole process repeats. It performs A* until CLOSED = {*g*, *f*, *h*, *e*, *i*}. The shortest path algorithm updates the heuristic values and the agent plans three actions: $g \rightarrow h \rightarrow i \rightarrow j$ (f) (it could also be $g \rightarrow$ f $\rightarrow$ e $\rightarrow d$). On the right hand graphs (g, h and i), we show our proposed scheme. Initially in (g), the agent is in state *l*. It performs A* until CLOSED = {*h*, *i*, *j*, *k*, *l*}. The shortest path algorithm on CLOSED causes no change. Then, we check if the final state where the agent is planning to move (state *g*) satisfies the updating condition.

Since $h(g)$ is going to change, we select the local space (I, F) around g with 5 states $I = \{g, f, h, e, i\}$ and $F = \{d, j\}$. We update I with the shortest path, obtaining the heuristic values of (e). Then, we plan a single action: $l \rightarrow k$. The whole process repeats with the agent in k. A* stops when CLOSED = $\{j, k\}$ because $h(j)$ satisfies the change condition. The local space (I, F) around j with 5 states is $I = \{j, k, i, l, h\}$ and $F = \{g, m\}$. The shortest path algorithm produces the changes reported in (i) and since a heuristic repair has been done, a single action is planned: $k \rightarrow l$.

LRTA*(k, d)

The strategy explained in the previous section is implemented in the LRTA*(k, d) algorithm. It is a LRTA*-based algorithm (Korf 1990), that propagates heuristic updates up to k states, following the bounded propagation idea of (Hernandez & Meseguer 2005) (Hernandez & Meseguer 2007). In addition, it is able to plan either 1 or up to d actions per planning step. It includes the following features:

- Lookahead using A*. Following the idea of (Koenig 2004), the lookahead required to plan more than one action per step is done using the well-known A* algorithm.
- Local space selection. When the heuristic value of a state x in the lookahead changes, the local space (I, F) around x (where 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).
- Propagation in local space. Once the local space (I, F) is selected, propagation of heuristic changes into the local space is done using the Dijkstra shortest paths algorithm, as done by (Koenig 2004).

LRTA*(k, d) is more that a novel combination of existing techniques. As a new element, the algorithm determines the number of actions to plan depending on the quality of the heuristic found in the lookahead. If the heuristic value of some state satisfies the updating condition, this change is propagated up to k states and only one action is planned only. If no heuristic value satisfies the updating condition in the lookahead states, a sequence of d actions are planned. These actions are executed in the execution phase, taking into account that if an obstacle is found, the execution stops and a new planning phase starts.

LRTA*(k, d) appears in Figure 3. The central procedure is LRTA*(k,d)-trial, that is executed once per trial until solution is found (**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 an state whose heuristic value satisfies the updating condition, (ii) it finds a state w such that $g(w) = d$, 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 whose heuristic value satisfies the updating condition, (ii) a state w such that $g(w) = d$, or (iii) a solution state. Observe that said *path* has at least one state x, and that the only state

```
procedure LRTA*-LS(k) (X, A, c, s, G, k, d)
 1  for each x ∈ X do h(x) ← h0(x);
 2  repeat
 3     LRTA*(k,d)-trial(X, A, c, s, G, k, d);
 4  until h does not change;

procedure LRTA*-LS-trial(X, A, c, s, G, k, d)
 1  x ← s;
 2  while x ∉ G do
 3     path ← A*(x, d, g);
 4     z ← last(path);
 5     if Changes?(z) then
 6        (I, F) ← SelectLS(z, k);
 7        Dijkstra(I, F);
 8        y ← argmin_{v∈Succ(x)} [h(v) + c(x, v)];
 9        execute(a ∈ A such that a = (x, y));
10     x ← y;
11  else
12     x ← extract-first (path);
13     while path ≠ ∅ k do
14        y ← extract-first (path);
15        execute(a ∈ A such that a = (x, y));
16        x ← y;

procedure SelectLS (x, k): pair of sets;
 1  Q ← ⟨x⟩; F ← ∅; I ← ∅; cont ← 0;
 2  while Q ≠ ∅ ∧ cont < k do
 3     v ← extract-first(Q);
 4     y ← argmin_{w∈Succ(v) ∧ w ∉ I} h(w) + c(v, w);
 5     if h(v) < h(y) + c(v, y) then
 6        I ←I ∪ {v};
 7        cont ← cont + 1;
 8        for each w ∈ Succ(v) do
 9           if w ∉ I ∧ w ∉ Q then Q ← add-
   last(Q,w);
10     else if I ≠ ∅ then F ←F ∪ {v};
11  if Q ≠ ∅ then F ←F ∪ Q;
12  return (I, F);

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

Figure 3: The LRTA*(k, d) algorithm.

that might change its heuristic value is *last*(*path*). If this state satisfies the updating condition (line 5), then this change is propagated: the local space is determined (line 6) and updated using the shortest path algorithm (line 7). Then, one action is planned (line 8), executed (line 9) and the loop iterates (line 10). If *last*(*path*) does not change its heuristic value, then up to d actions are planned and executed (lines 12-16).

Function SelectLS(k, d) computes the local space (I, F) around x that has at most k states in I. The set F surrounds I immediately and completely. This function works as follows. It keeps a queue Q of the 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 in I, controlled by the counter *cont*. Then, a loop is executed until Q contains no elements or *cont* is equal to k (**while** loop, line 2). The first state v in Q is extracted (line 3). The state $y \leftarrow \arg\min_{w \in Succ(v) \wedge w \notin I} [c(v, w) + h(w)]$ is computed (line 4). If state v is going to change (line 5), it enters I and the counter is incremented (lines 6-7). Those successors of v which are not in I or Q enter Q by the rear (lines 8-9). Otherwise, v enters F (line 10). When exiting the loop, if Q still contains states, these are added to F.

Function Changes?(x) returns true if x satisfies the updating condition. Otherwise, it returns false (lines 2-3).

Since the heuristic always increases, LRTA(k,d) completeness is guaranteed (Theorem 1 of (Korf 1990) holds). If the heuristic is initially admissible, updating the local space with the shortest path algorithm keeps admissibility (Koenig 2004), so convergence to the optimal paths is guaranteed in the same terms as LRTA* (Theorem 3 of (Korf 1990) holds). So LRTA(k, d) inherits the good properties of LRTA*.

One might expect that LRTA*(k, d) collapses into LRTA*$_{LS}$(k) when d = 1. However, this is not the case. When d = 1, these two algorithms basically differ in the following. If the heuristic of the current state satisfies the updating condition, LRTA*$_{LS}$(k) updates it and propagates this change in a local space constructed around the current state. In this case, LRTA*(k, 1) behaves exactly like LRTA*$_{LS}$(k). But if the heuristic of the current state does not change, LRTA*(k, 1) generates a local space using the A* algorithm, and if the heuristic of some state of this local space satisfies the updating condition, it is updated and this change is propagated in a local space around that state.

Experimental Results

We compare the performance of LRTA*(k, d) with LRTA* (version of Koenig), RTAA* and LRTS(γ=1, T=∞). Parameter k is the size of the local space, where bounded propagation is performed; it is usually taken as the lookahead parameter for LRTA* and RTAA*. We have used the values k = 5, 10, 20, 40, 80. For LRTS, we have used the following values for the lookahead depth: 2, 3, 4, 6 (corresponding to local spaces of size close to those generated by the four last k values). Parameter d is the upper limit on the number of planned actions per step for LRTA*(k, d). We have used the values d = 1, 2, 4, 6.

Benchmarks are 4-connected grids and five different maps modeled from a role-playing game (BioWare Corp. 1998). For grids, we use Manhattan distance as the initial heuristic. We use the following grids: **Grid35**. Grids of size 301 × 301 with a 35% of obstacles placed randomly. Here, Manhattan distance tends to provide a reasonably good advice. **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. The number of states in each computer game map is 2,765, 7,637, 13,765, 14,098, and 16,142. The maps are 8-connected worlds. We use the maximum of the absolute differences of the x and y coordinates of two cells as the initial heuristic. In all benchmarks, the starting 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. Results consider the first trial and the convergence to optimal trajectories. We present the solution cost (= number of actions performed to reach the goal) and total search time (until the agent reaches the goal) in milliseconds, plotted against k, averaged over 1,500 different instances in grids, and over 10,000 instances in maps (2,000 instances for each map).

LRTA*, RTAA* and LRTA*(k, d) results appear in Figure 4, while LRTS(γ=1, T=∞) results appear in Table 1.They are not included in Figure 4 for clarity purposes. Solution costs are always worse than those provided by other algorithms and in some cases their inclusion caused to change the plot scale. From this point on, we limit the discussion to LRTA*, RTAA* and LRTA*(k, d).

Results for first-trial on Grid35 appear in the first row of Figure 4. We observe that the solution cost decreases monotonically as k increases, and for LRTA*(k, d) it also decreases monotonically as d increases. LRTA*(k, d) versions obtain the best results for low lookahead, and all algorithms have a very similar cost for high lookahead (for k = 80 LRTA* obtains the minimum cost, followed closely by RTAA* and LRTA*(80,6)). Considering total search time, RTAA* and LRTA* increase monotonically with k, something which does not occur for LRTA*(k, d). This algorithm (especially d = 1, 2 although the four d values have a similar behavior) starts at the same level as the other algorithms, decreases slightly with k and after k = 20, increases very slightly. Globally, it remains fairly stable. For medium and high lookahead, its total search time is much shorter than RTAA* and LRTA*. So, in the whole lookahead range, LRTA*(k, d) offers a very good performance in solution cost with a low (often the lowest) total search time.

Results for the first-trial on Maze appear in the second row of Figure 4. Regarding the solution cost, the pattern is similar to the observed in Grid35: it decreases monotonically as k increases, LRTA*(k, d) versions obtain lower costs than RTAA* and LRTA* (except for k = 5 and d = 1) and their costs decrease as d increases. All algorithms have a similar cost with k = 80 (LRTA*(80, 2) reaches the minimum cost, followed by LRTA*(80, 4) and LRTA*(80, 1)). Regarding total search time, the pattern is different from Grid35. All algorithms (except LRTA(5, 1)) require a similar time for k = 5. Then, the time required by all algorithms decreases as k increases until some k value where the time increases again. The interesting point is that LRTA* and RTAA* start increasing for a relatively low lookahead (k = 10, 20) while LRTA(k, d) starts increasing for larger lookahead (k = 40). In addition, this increment is very low. As a result, all versions of LRTA*(k, d) offer the shortest search times for medium to high lookahead. Again, in the whole range of lookahead, LRTA*(k, d)

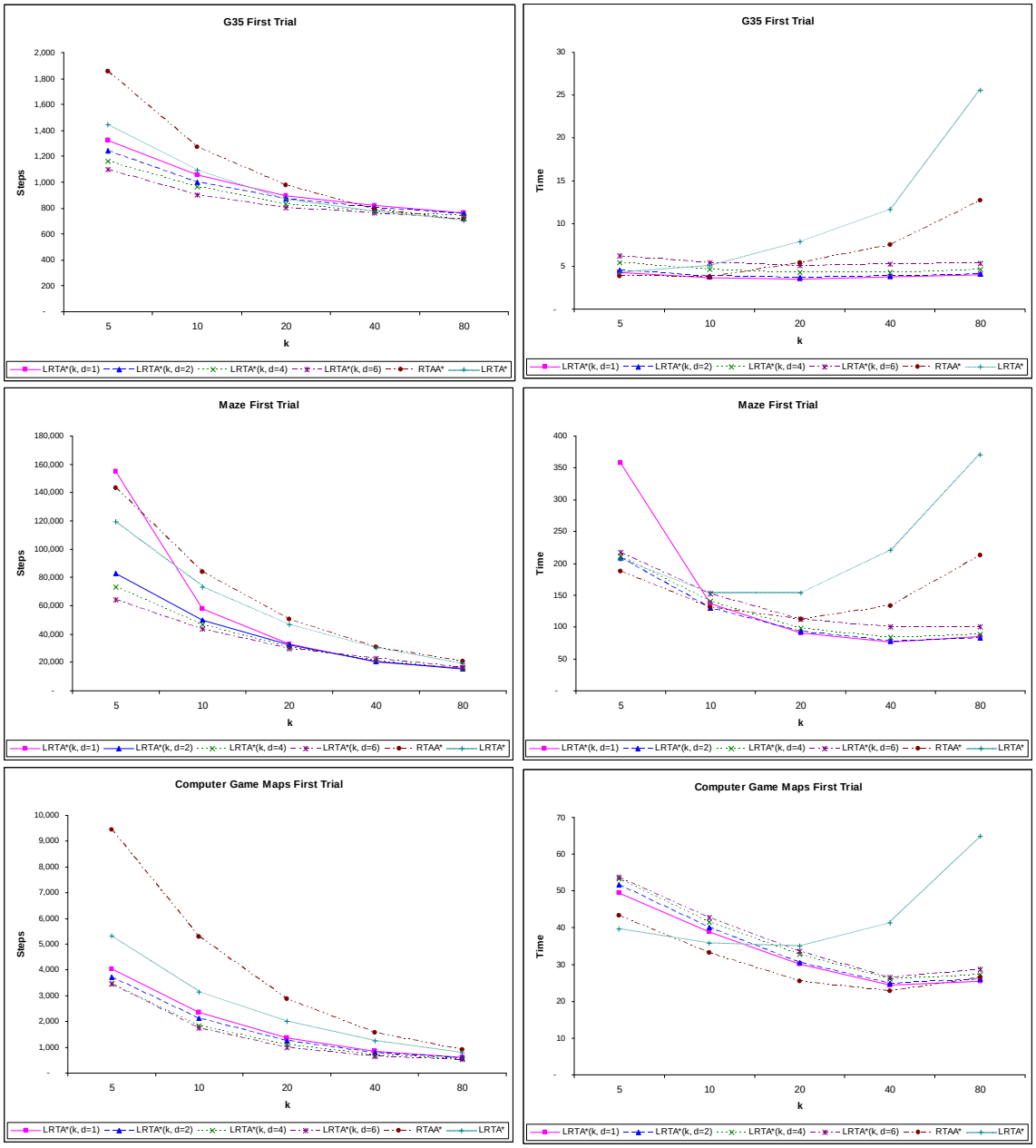

Figure 4: Experimental results on Grid35, Maze and game maps. We present solution cost and total search time for the first trial.

offers a very good performance in solution cost with a low (often the lowest) total search time.

Results for the first-trial on computer game maps appear in the third row of Figure 4. Regarding the solution cost and total search time, the pattern is similar to the observed in Grids: the cost decreases monotonically as k increases and LRTA*(k, d) versions obtain lower costs than RTAA* and LRTA*. The total search time required by all algorithms decreases as k increases until some k value where the time increases again. LRTA* starts increasing for relatively low lookahead (k = 10) while RTAA* and LRTA*(k, d) starts increasing for intermediate values of lookahead (k = 40).

Results for convergence on Grid35 appear in the first row of Figure 5. Solution cost results are similar to the first trial on the same benchmark. Regarding total search time, the decreasing-increasing curve observed in the Maze benchmark appears for LRTA* (inflexion point at k = 20) and RTAA* (inflexion point at k = 40). Interestingly, LRTA*(k, d) versions keep decreasing. So here LRTA*(k, d) obtains the minimum cost and the shortest search time at the maximum lookahead tested (k = 80). Differences among d values are small.

Results for convergence on Maze appear in the second row of Figure 5. For the solution cost, the pattern already observed in the previous experiments is repeated here.

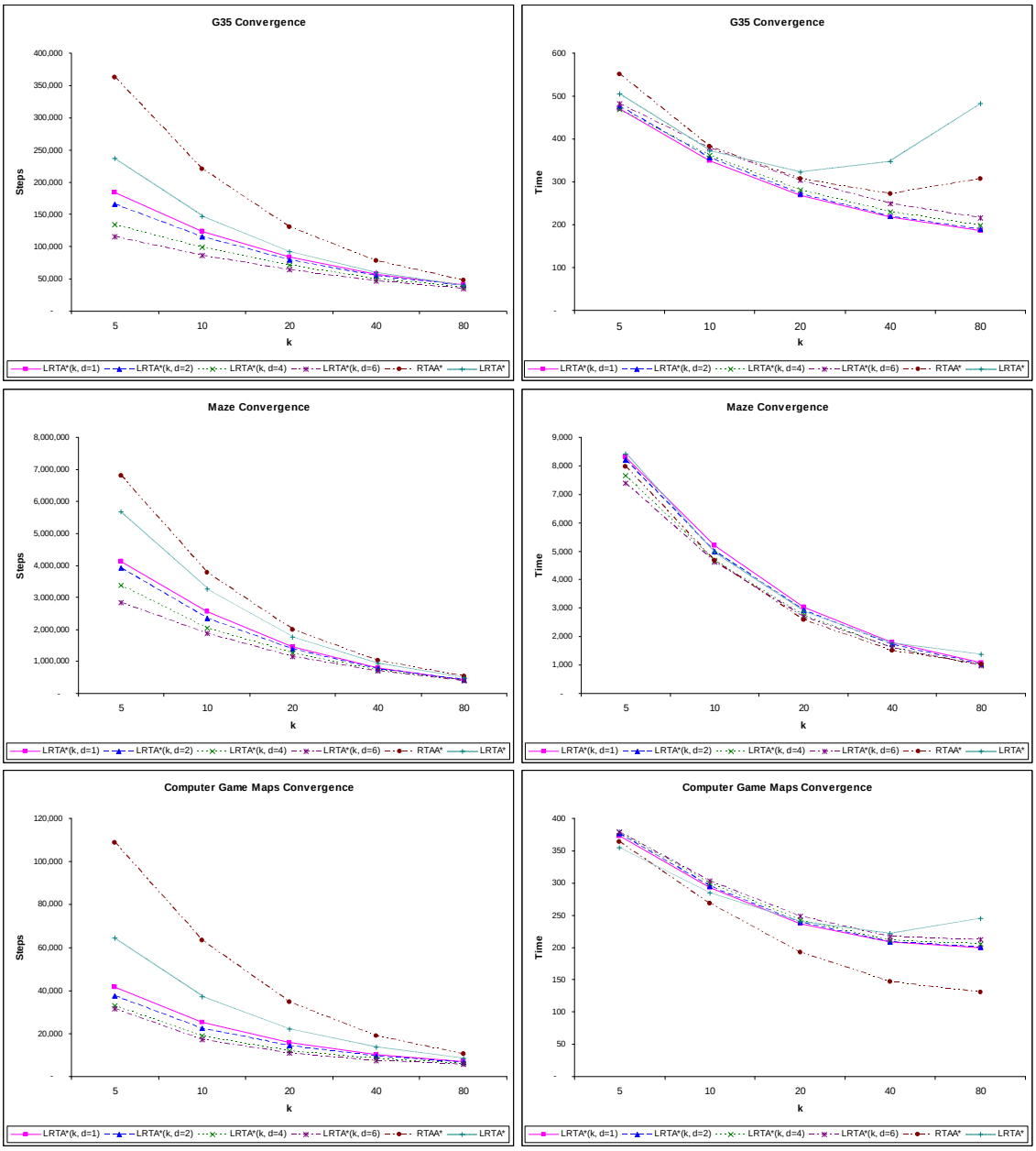

Figure 5: Experimental results on Grid35, Maze and game maps. We present solution cost and total search time for convergence.

Regarding total search time, all algorithms show a similar behavior: for each k value, all have similar time requirements, which decrease monotonically as k increases. Therefore, for low lookahead LRTA*(k, d) offers the minimum cost, while there is little difference for high lookahead.

Results for convergence on computer game maps appear in the third row of Figure 5. For the solution cost, the pattern already observed in the previous experiments is repeated here. Regarding total search time, the decreasing-increasing curve observed in the Maze benchmark appears for LRTA* (inflexion point at k = 40). Interestingly, LRTA*(k, d) versions and RTAA* keep decreasing.

Summarizing, there is a common pattern in solution cost: all algorithms improve as lookahead increases, for low and medium lookahead LRTA*(k, d) versions offer the best cost, while for high lookahead all algorithm achieve a similar cost. Regarding the total search time, we observe some differences. Considering four experiments (first trial in Grid35, Maze and computer game maps, convergence in Grid35) all algorithms require a similar time for low lookahead. This time increases drastically for RTAA* (except in computer game maps) and LRTA* for high lookahead, while it remains much lower for LRTA*(k, d). Regarding convergence in Maze and computer game maps, for all values of lookahead tested all algorithms require a similar time (except RTAA* in computer game maps, for k >20, the required time is smaller) which decreases monotonically as lookahead increases. In the whole range of lookahead values, LRTA*(k, d) offers a very good

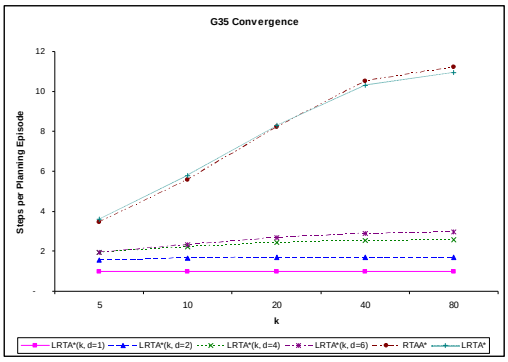

Figure 6. Number of actual moves per planning episode.

(often the minimum) solution cost with a low (often the lowest) total search time. With high lookahead (the area where solutions of lowest cost are found), RTAA* and LRTA* search times are much higher than LRTA*(k, d) (up to two times for RTAA* and up to five times for LRTA*), except for convergence in Maze, where differences are small (nevertheless, LRTA*(80, 6) is the fastest algorithm), and for convergence in computer game maps, where RTAA* is the fastest algorithm. Regarding parameter d, low values (d = 1, 2) achieve the shortest search times without a substantial decrement in solution cost.

From these results, it is clear that the crucial parameter is lookahead. If k can be made large enough, then all algorithms achieve a similar solution cost, so it is meaningful to select LRTA*(k, d) which offers the shortest search times. If k has to be small or medium, LRTA*(k, d) offers the best solution cost graded by parameter d, at low search time.

It is interesting to know the actual number of moves per step performed by the algorithms. In Figure 6, it appears the average number of actual moves per planning episode against lookahead for convergence in Grid35 (for Maze and computer game maps results are similar). The number of actual moves increases as k increases. There are two groups of algorithms. On one hand, RTAA* and LRTA* (which perform almost exactly the same number of moves) increase steeply with k (in fact, they have k as the upper limit of planned moves). On the other hand, LRTA*(k, d) versions increase smoothly, they are almost flat. RTAA* and LRTA* perform much more moves than LRTA*(k, d) but many of them are not good (only in this way their worse performance can be explained). The quality of each move is crucial here (a move in the wrong direction requires another restoring move, and moves have some cost). LRTA*(k, d) performs a lower number of better moves, offering the best performance.

We have compared two single-move algorithms, LRTA*$_{LS}$(k) and LRTA*(k, 1). Their differences were previously indicated in the LRTA*(k, d) section (the plots are not including, for space reasons). LRTA*(k, 1) obtains better solution costs than LRTA*$_{LS}$(k), with lower search times. Since their basic difference is the repair strategy of heuristic inaccuracies (LRTA*$_{LS}$(k) considers the current state only, while LRTA*(k, 1) considers any state of the local space), this plot shows that repairing heuristic inaccuracies of states of the local space as soon as they are detected, is a good strategy that brings substantial benefits in the long term. This results allow us to infer that LRTA*(k, d) improves LRTA*$_{LS}$(k) for d > 1.

	First Trial				Convergence			
	Grid35		Maze		Grid35		Maze	
	Cost	Time	Cost	Time	Cost	Time	Cost	Time
2	2,997.0	5.3	286,298.0	384.8	454,010.1	812.3	9,342,177.6	12,495.8
3	3,406.2	7.0	318,744.6	411.0	625,132.8	1,249.4	13,484,604.6	17,103.7
4	2,838.7	6.9	346,511.8	439.7	473,503.1	1,059.6	6,213,024.9	7,692.1
6	2,783.7	9.7	349,338.4	445.0	626,929.4	1,739.0	6,925,467.7	8,431.9

Table 1: LRTS(γ=1, T=∞) results. The lookahead depth appears in the leftmost column.

Conclusions

We have presented LRTA*(k, d), a new real-time algorithm able to plan several moves per planning step. It includes features already introduced in real-time search (A*, Dijkstra shortest path, bounded propagation). In addition, it implements two main ideas (i) the heuristic quality has to be taken into account when planning several moves (in the current version, it allows for planning several moves when no heuristic inaccuracy is detected), and (ii) as soon as a heuristic inaccuracy is detected, it must be repaired, no matter if it is located at the current state or not. LRTA*(k, d) is complete and converges to optimal paths after repeated executions on the same instance. Experimentally, we have seen on three Benchmarks that LRTA*(k, d) outperforms LRTA* (version of Koenig), RTAA* and LRTS (γ=1, T=∞).

Acknowledgments

We would like to thank Vadim Bulitko from the University of Alberta for his useful comments.

References

BioWare Corp. 1998. Baldur's Gate. *Interplay*. http://www.bioware.com/bgate/

Bulitko, V., and Lee, G. 2006. Learning in real time search: a unifying framework. *JAIR* 25:119–157.

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

Hernandez, C., and Meseguer, P. 2005. Lrta*(k). *In Proceedings of the 19th IJCAI*, 1238–1243.

Hernandez, C., and Meseguer, P. 2007. Improving lrta*(k). *In Proceedings of the 20th IJCAI*, 2312–2317.

Koenig, S., and Likhachev, M. 2002. D*lite. *In Proceedings of the AAAI*, 476–483.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive a*. *In Proceedings of the AAMAS*, 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 AAMAS*, 864–871.

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