Real-Time Heuristic Search with Depression Avoidance

Carlos Hernández
Departamento de Ingeniería Informática
Universidad Católica de la Ssma. Concepción
Concepción, Chile

Jorge A. Baier
Departamento de Ciencia de la Computación
Pontificia Universidad Católica de Chile
Santiago, Chile

Abstract

Heuristics used for solving hard real-time search problems have regions with depressions. Such regions are bounded areas of the search space in which the heuristic function is exceedingly low compared to the actual cost to reach a solution. Real-time search algorithms easily become trapped in those regions since the heuristic values of states in them may need to be updated multiple times, which results in costly solutions. State-of-the-art real-time search algorithms like LSS-LRTA*, LRTA$^*(k)$, etc., improve LRTA*'s mechanism to update the heuristic, resulting in improved performance. Those algorithms, however, do not guide search towards avoiding or escaping depressed regions. This paper presents *depression avoidance*, a simple real-time search principle to guide search towards avoiding states that have been marked as part of a heuristic depression. We apply the principle to LSS-LRTA* producing aLSS-LRTA*, a new real-time search algorithm whose search is guided towards exiting regions with heuristic depressions. We show our algorithm outperforms LSS-LRTA* in standard real-time benchmarks. In addition we prove aLSS-LRTA* has most of the good theoretical properties of LSS-LRTA*.

1 Introduction

In many real-world applications, agents need to act quickly in dynamic, initially unknown domains. Example applications range from robot navigation, to agent navigation in games (e.g., Baldur's Gate, Starcraft, etc.). Indeed, the computer game company Bioware imposes a time limit of 1-3 milliseconds for each search episode [Bulitko *et al.*, 2007].

Real-time search (e.g., [Korf, 1990; Koenig, 2001]) is a standard paradigm for solving search problems in those settings. Instead of running a computationally expensive procedure to generate a conditional plan at the outset, real-time algorithms interleave planning and execution. As such, they usually run a computationally cheap observe-plan-act cycle, in which the environment is observed, an action is selected, and then executed. Like standard A* search [Hart *et al.*, 1968], they use a heuristic function to guide action selection. However, as the environment is unveiled, the agent updates its internal belief about the structure of the search space, updating (i.e. *learning*) the heuristic value for some states. The observe-plan-act cycle is executed until a solution is found.

Early heuristic real-time algorithms like LRTA* and RTA* [Korf, 1990] perform poorly in presence of *heuristic depressions* [Ishida, 1992]. Intuitively, a heuristic depression is a bounded region of the search space in which the heuristic is unrealistic with respect to the heuristic value of the states in the border of the region. When visiting states with heuristic depressions, LRTA* and RTA* usually become "trapped". To exit these regions they will visit most states in the depression, possibly many times.

State-of-the-art heuristic real-time search algorithms are able to escape heuristic depressions more quickly than LRTA* or RTA*. They do so by performing more *search*, more *learning* or a combination of both. More search typically involves selecting an action by looking farther away in the search space; thus, we say these algorithms perform *lookahead* search. More learning involves updating the heuristic values of several states in a single iteration. There are many algorithms that use one or a combination of these techniques (e.g. [Bulitko and Lee, 2006; Koenig and Likhachev, 2006; Björnsson *et al.*, 2009; Hernández and Meseguer, 2005; 2007; Koenig and Sun, 2009]).

LSS-LRTA* [Koenig and Sun, 2009] is a state-of-the-art algorithm that generalizes LRTA* and does both lookahead search and learning, that has been shown to perform very well in practice. However, despite the use of more elaborate techniques, it may still perform poorly in presence of heuristic depressions. This is because the main mechanism to escape depressions remains the same as in early algorithms: to increase the heuristic value of states inside the depression.

In this paper, we propose an approach to *actively* guide search towards avoiding heuristically depressed regions that we call *depression avoidance*. To apply this approach to a real-time search algorithm we need to define: (1) a mechanism for detecting states that belong to a heuristic depression, and (2) a strategy that leads the agent to prefer to move to states that are *not* identified as being in a heuristic depression. We apply this approach to the state-of-the-art algorithm LSS-LRTA*, producing a new algorithm, aLSS-LRTA*.

We evaluate aLSS-LRTA* on standard benchmarks and shows that, although simple, our modifications to LSS-

LRTA* yield significant performance improvements. On average, for an equal amount of lookahead aLSS-LRTA* finds better solutions in less time. Additionally, we perform a theoretical analysis showing that desirable properties, such as termination, hold for aLSS-LRTA*.

The paper is organized as follows. We explain basic concepts of real-time search in the next section. We continue elaborating on the concept of heuristic depression. We then describe our algorithm in detail and continue with a theoretical and experimental analysis. Finally, we briefly sketch our conclusions.

2 Preliminaries

A search problem P is a tuple (S, A, c, s_0, G), where (S, A) is a digraph that represents the search space. The set S represents the *states* and the arcs in A represent all available actions. A does not contain elements of form (x, x). In addition, the cost function $c : A \mapsto \mathbb{R}^+$ associates a cost to each of the available actions. Finally, $s_0 \in S$ is the start state, and $G \subseteq S$ is a set of goal states. We say that a search space is *undirected* if whenever (u, v) is in A then so is (v, u). We assume that in undirected spaces $c(u, v) = c(v, u)$, for all $(u, v) \in A$. The successors of a state u are defined by $Succ(u) = \{v \mid (u, v) \in A\}$. Two states are *neighbors* if they are successors of each other. A heuristic function $h : S \mapsto [0, \infty)$ associates to each state s an approximation $h(s)$ of the cost of an path from s to a goal state. h is *consistent* iff $h(g) = 0$ for all $g \in G$ and $h(s) \leq c(s, w) + h(w)$ for all states $w \in Succ(s)$. We refer to $h(s)$ as the h-value of s. We assume familiarity with the A* algorithm [Hart *et al.*, 1968]: $g(s)$ denotes the cost of the path from the start state to s, and $f(s)$ is defined as $g(s) + h(s)$. The f-value and g-value of s refer to $f(s)$ and $g(s)$ respectively.

2.1 Real-Time Search

The objective of a real-time search algorithm is to make an agent travel from an initial state to a goal state performing, between moves, an amount of computation bounded by a constant. An example situation is pathfinding in previously unknown grid-like environments. There the agent has memory capable of storing its current belief about the structure of the search space, which it initially regards as obstacle-free (this is usually referred to as the *free-space assumption* [Koenig *et al.*, 2003]). The agent is capable of a limited form of sensing: only obstacles in the neighbor cells can be detected. When obstacles are detected, the agent updates its map accordingly.

Local Search Space LRTA* (LSS-LRTA*) [Koenig and Sun, 2009] (Algorithm 1) is a state-of-the-art real-time heuristic search algorithm which generalizes LRTA* [Korf, 1990]. It iteratively executes an lookahead-update-act cycle until the goal is reached. The procedure has three local variables. Variable s stores the current position of the agent. Variable $c(s, s')$ contains the cost of moving state s to a successor s'. Variable h is such that $h(s)$ contains the heuristic value for s. All three variables change over time. Initially c is such that no obstacles are assumed; i.e., $c(s, s') < \infty$ for any two neighbor states s, s'. The initial value of h is given as a parameter.

In the lookahead phase (Line 3–5), it determines where to proceed next. To perform this phase, it uses A* search to select the best node to go. The search however, is restricted to expand at most k nodes. The number k is called *lookahead* parameter, and the set of states generated by A* are referred to as the *local search space*. In the update phase (Line 6), it updates (raises) the heuristic value of states in the local search space to the maximum possible value that keeps the heuristic consistent with respect to the other nodes in the frontier of the local search space. LSS-LRTA* achieves this by a call to a version of Dijsktra's algorithm. Finally, in the acting phase (Line 7) it moves the agent to the final position, updating the cost variable c if obstacles are observed.

LSS-LRTA* is equivalent to LRTA* when the number of nodes expanded by A* is limited to 1 (i.e., only the current state is expanded). Koenig and Sun [2009] show that, in several benchmarks, the quality of the solutions returned improve as the lookahead parameter increases. Nevertheless, they show that total search time increases as more lookahead is performed and that LSS-LRTA* outperforms incremental algorithms like D* Lite when under time pressure.

Algorithm 1: Pseudo-code for LSS-LRTA*

```
1 s ← s_0
2 while s ∉ G do
3     Expand at most k states with A*
4     if the open list is empty then return no-solution
5     y ← Best state in the open list
6     Update the heuristics of states in the Closed list
7     Move the agent from s to y through the optimal path identified by
      A*. Stop if an action cost along the path is updated (to infinity).
8     s ← current agent position
9     Update action costs (if they have increased)
```

3 Heuristic Depressions

In real-time search problems, however, heuristics usually contain *depressions*. Identification of these depressions is central to our algorithm.

Intuitively, a heuristic depression is a bounded region of the search space containing states whose heuristic value is too low with respect to the heuristic values of states in the border of the depression. Depressions exist naturally in heuristics used along with real-time heuristic search algorithms and are also generated during runtime. As we have seen above, heuristic real-time algorithms build solutions incrementally, incrementing the heuristic values associated to certain states as more information is gathered from the environment.

Ishida [1992] gave a constructive definition for heuristic depression. The construction starts with a node s such that its heuristic value is equal to or less than those of the surrounding states. The region is then extended by adding a state of its border if all states in the resulting region have a heuristic value lower or equal than those of the states in the border. As a result, the heuristic depression D is a maximal connected component of states such that all states in the boundary of D have a heuristic value that is greater than or equal to the heuristic value of any state in D.

It is known that LRTA* behaves poorly in presence of heuristic depressions [Ishida, 1992]. To see this, assume that

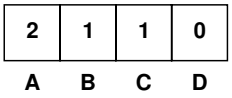

Figure 1: A 4-state grid-like search space with unitary costs. Cells show their h-value. **D** is the goal state.

LRTA* visits a state in a depression and that the solution node lies outside the depression. To exit the depressed region the agent must follow a path in the interior of the depressed region, say, $s_1 \dots s_n$, finally choosing a state in the border of the region, say s_e. While visiting s_n the agent chooses s_e as the next move, which means that s_e *minimizes* the estimated cost to reach a solution among all the neighbors of s_n. In problems with uniform action costs, this can only happen if $h(s_e)$ is *lower or equal* than the heuristic value of all other neighbors of s_n. This fact actually means that the depression in that region of the search space no longer exists, which can only happen if the heuristic values of states in the originally depressed region have been updated (increased). For LRTA*, the update process may be quite costly: in the worst case, indeed, *all* states in the depression may need to be updated and each state may need to be updated several times.

Ishida's definition is, nonetheless, restrictive. In fact, it does not take into account the *costs* of the actions needed to move from the interior of the depression to the exterior. A closed region of states may have unrealistically low heuristic values even though the heuristic values in the interior are greater than the ones in the border. We propose a more intuitive notion of depression when costs are taken into account. The formal definition follows.

Definition 1 (Cost-sensitive heuristic depression) *A connected component of states D is a cost-sensitive heuristic depression of a heuristic h iff for any state $s \in D$ and every state s' in the boundary of D, $h(s) < k(s, s') + h(s')$, where $k(s, s')$ denotes the cost of the cheapest path that starts in s, traverses states only in D, and ends in s'.*

Figure 1 shows an example search space in which our definition for cost-sensitive depression defers from Ishida's. Indeed, no matter what "start state" is used with Ishida's procedure, A is never part of a heuristic depression. On the other hand $\{A, B\}$ *is* a cost-sensitive depression. The fact that A is in a cost-sensitive depression is important because this means that the h-value of A is low with respect to the border cell C. Indeed, intuitively if it takes two actions to get to state C, then the heuristic value of A should be 3 instead of 2.

4 Depression Avoidance

As seen above, a major issue in solving real-time search problems is the presence of heuristic depressions. Algorithms like LSS-LRTA*, and LRTA$^*(k)$ escape those regions faster than LRTA* because they increment the heuristic value of more states in every iteration. Thus, if a set of states D is a heuristic depression, and the current position of the agent is in D, LSS-LRTA*, LRTA$^*(k)$ and LRTA* exit the region given by D when the heuristic value of those states has increased sufficiently so that the depression in D has disappeared. LSS-LRTA* exits the depressions more quickly than LRTA* both because the heuristic function increases for states in D more quickly and because of its ability to do lookahead.

To escape heuristic depressions we propose a different strategy: *depression avoidance*, which guides search for a quick way out of depressions by explicitly avoiding them. Depression avoidance is a very simple principle that dictates that search should be guided away from states identified as being in a heuristic depression. There are many ways in which one could conceive the implementation of this principle in a real-time heuristic search algorithm. Below, we propose one of the many possible realizations of the principle within the state-of-the-art LSS-LRTA* algorithm. The result is aLSS-LRTA*, an algorithm based on the principle that has good theoretical properties.

aLSS-LRTA* is a simple yet effective modification of LSS-LRTA*. The main conceptual difference between aLSS-LRTA* and LSS-LRTA* is that the former prefers to move to states that are not yet identified as belonging to a depression. To achieve this, we modify two aspects of LSS-LRTA*. First, we modify the procedure that LSS-LRTA* uses to update the heuristic values to *mark* the states that are inside depression. Second, we modify the mechanism to select at what state the agent will attempt to move to. To select the next move, the algorithm chooses the best state generated by the lookahead phase that has *not* been marked as in a depression. If such a state does not exist the algorithm selects the best state seen during the lookahead phase, just like LSS-LRTA* would do.

Details of the pseudo code for aLSS-LRTA* are shown in Algorithm 2. The lookahead procedure (Line 6) is exactly the same used by LSS-LRTA*: a standard A* search that will expand at most k nodes and, additionally, stops if the goal state is inserted into $Open$. The set of states that were expanded, as usual, are stored in $Closed$.

The heuristic values of states expanded by the lookahead search will be updated by a call to our modified Dijkstra procedure. In a nutshell, this procedure changes the values of the heuristic for all states in $Closed$ to the highest possible value that guarantees the consistency of the heuristic. The main difference between our procedure and that of LSS-LRTA* is Line 32, which does not exist in LSS-LRTA*. Here, for a state s in the search space, we set s.updated to true if the heuristic value for h changes as a result of the update process. In the following section, we formally prove that this means that s was inside a heuristic depression (Theorem 1).

Finally, in Line 8, the algorithm chooses the next state to go to by extracting the state with lowest f-value in $Open$ which is not marked. If no unmarked states exist, the algorithm chooses the state with best f-value in $Open$.

An Example Fig. 2 shows an example that illustrates the difference between LSS-LRTA* and aLSS-LRTA* with lookahead parameter equal to two. After 3 search episodes, we observe that aLSS-LRTA* avoids the depression leading the agent to a position 2 steps closer to the goal than LSS-LRTA*.

5 Evaluation

Below we evaluate aLSS-LRTA* both theoretically and experimentally.

Algorithm 2: aLSS-LRTA*: Learning Real-Time A* with propagation and depression avoidance.

```
   Input: A search problem P, a heuristic function h, a cost function c.
1  for each s ∈ S do
2      s.updated ← false
3      h_0(s) ← h(s)
4  s_current ← s_0
5  while s_current ∉ G do
6      A* ()
7      if Open = ∅ then return no-solution
8      s_next ← Extract-Best-State()
9      Dijkstra ()
10     move the agent from s_current to s_next through the path
       identified by A*. Stop if an action cost along the path is updated
       (to infinity).
11     s_current ← current agent position
12     update action costs (if they have increased)
13 procedure A* ()
14     for each s ∈ S do g(s) ← ∞
15     g(s_current) ← 0
16     Open ← ∅
17     Insert s_current into Open
18     expansions ← 0
19     while each s' ∈ Open with minimum f-value is such that s' ∉ G
       and expansions < k do
20         s ← argmin_{s'∈Open} g(s') + h(s')
21         Insert s into Closed
22         for each s' ∈ Succ(s) do
23             if g(s') > g(s) + c(s, s') then
24                 g(s') ← g(s) + c(s, s')
25                 s'.back = s
26                 Insert s' in Open
27         expansions ← expansions + 1
28 procedure Dijkstra ()
29     for each s ∈ Closed do h(s) ← ∞
30     while Closed ≠ ∅ do
31         Extract an s with minimum h-value from Open
32         if h(s) > h_0(s) then s.updated = true
33         if s ∈ Closed then delete s from Closed
34         for each s' such that s ∈ Succ(s') do
35             if s' ∈ Closed ∧ h(s') > c(s', s) + h(s) then
36                 h(s') ← c(s', s) + h(s)
37                 if s' ∉ Open then Insert s' in Open
38 procedure Extract-Best-State ()
39     if Open contains an s such that s.updated = false then
40         s ← argmin_{s'∈Open∧s'.updated=false} g(s') + h(s')
41     else
42         s ← argmin_{s'∈Open} g(s') + h(s')
43     return s
```

5.1 Theoretical Analysis

In this section we show that aLSS-LRTA* satisfies most of the interesting properties of LSS-LRTA*. Specifically, we can use the same proofs by Koenig and Sun [2009] to prove the heuristic h is non-decreasing and remains consistent if the provided heuristic is consistent. Below we present an analysis of some properties specific to aLSS-LRTA*. We establish some results that also hold for aLSS-LRTA*, but that need different proofs than those known for LSS-LRTA*.

Before stating the results we introduce some useful notation. We denote the heuristic value function h in aLSS-LRTA* at the beginning of the n-th iteration as h_n. As such,

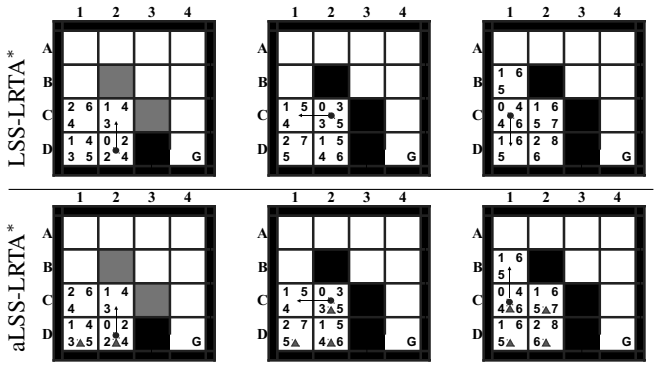

Figure 2: First 3 search episodes of LSS-LRTA* (above) and aLSS-LRTA* (below) with lookahead equal to 2 in a 4-connected grid world with action unitary action costs, where the initial state is D2, and the goal is D4. Numbers in cell corners denote the g-value (upper left), f-value (upper right), h-value (lower left), and new h-value of an expanded cell after an update (lower right). Only cells that have been in a closed list show four numbers. Cells just generated by A* or (i.e., in their $Open$ lists) show three numbers, since their h-values have not been updated. Triangles (▲) denote states with updated flag set to true after the search episode. The heuristic used is the manhattan distance. We assume ties are broken by choosing first right then bottom then left then top adjacent cell. The position of the agent is given by the dot. A grid cell is shaded (gray) if it is a blocked cell that the agent has not sensed yet. A grid cell is black if it is a blocked cell that the agent has already sensed. The best state chosen to move the agent to after lookahead is pointed by an arrow.

h_{n+1} represents the result of updating h_n by the call in Line 9. Likewise, we denote the value of the cost function c in aLSS-LRTA* at the beginning of the n-th iteration by c_n. Finally, $k_n(s, s')$ denotes the cost of the cheapest path from s to s' that traverses states only in $Closed$ until it reaches s', with respect to the cost function c_n.

The proposition below gives an expression h-value of a state in $Closed$ after the call to the Dijkstra algorithm. Intuitively, the heuristic is updated soundly with respect to the boundary.

Proposition 1 *Let s be a state in $Closed$ right after the call to A* in the n-th iteration of the algorithm. Then,*

$$h_{n+1}(s) = \min_{s_b \in Open} k_n(s, s_b) + h_n(s_b).$$

The updated flag is set to true as soon as the heuristic has been updated:

Proposition 2 *If $h_{n+1}(s) > h_n(s)$ then $s.\mathit{updated} = true$ after the n-th iteration of the algorithm.*

The following theorem establishes that whenever a state s is marked as updated, then s belongs to a heuristic depression.

Theorem 1 *Let s be a state such that $s.\mathit{updated}$ switches from false to true between iterations n and $n+1$. Then s is in a cost-sensitive heuristic depression of h_n.*

Proof: Let D be the maximal connected component of states connected to s such that (1) all states in D are in $Closed$ after the call to A* in iteration n, and (2) any state s_d in D is such that $h_{n+1}(s_d) > h_n(s_d)$. We prove that D is a cost-sensitive heuristic depression of h_n.

Let s' be a state in the boundary of D. We first show that $h_n(s') = h_{n+1}(s')$. Indeed, $s' \in Closed \cup Open$. Assume $s' \in Closed$. Then, since $s' \notin D$, it must be because it does not satisfy condition (2) of the definition of D, and hence $h_{n+1}(s') \leq h_n(s')$. However, since the heuristic is non-decreasing, it must be that $h_n(s') = h_{n+1}(s')$. On the other hand, if s' is in $Open$, its heuristic value is not changed and thus also $h_n(s') = h_{n+1}(s')$.

Let s_d be a state in D. We continue the proof by showing that $h_n(s_d)$ is too low with respect to $h_n(s')$, which means that D is a heuristic depression of h_n. We distinguish two cases: if $s' \in Closed$,

$$h_n(s') = k_n(s', s_b) + h_n(s_b), \tag{1}$$

for some $s_b \in Open$. On the other hand, since the heuristic value has increased for s_d, $h_n(s_d) < h_{n+1}(s_d) = \min_{s'_b \in Open} k_n(s_d, s'_b) + h(s'_b)$; in particular, $h_n(s_d) < k_n(s_d, s_b) + h_n(s_b)$. Since $k_n(s_d, s_b)$ is the optimal cost to go from s_d to s_b, $k_n(s_d, s_b) \leq k_n(s_d, s') + k_n(s', s_b)$. Substituting $k_n(s_d, s_b)$ in the previous inequality we have

$$h_n(s_d) < k_n(s_d, s') + k_n(s', s_b) + h_n(s_b). \tag{2}$$

We now substitute the right-hand side of (2) using (1), and we obtain $h_n(s_d) < k_n(s_d, s') + h_n(s')$. Finally, if $s' \in Open$ by Proposition 1 and the fact that $h_{n+1}(s_d) > h_n(s_d)$, we also have that $h_n(s_d) < k_n(s_d, s') + h_n(s')$. This proves that s_d is in a cost-sensitive heuristic depression of h_n. ■

Finally, we prove that the algorithm leads an agent to a solution if such a solution exists when the heuristic is consistent.

Theorem 2 *Let P be an undirected finite real-time search problem such that a solution exists. Let h be a consistent heuristic for P. Then aLSS-LRTA*, used with h, will find a solution for P.*

5.2 Experimental Evaluation

We compared aLSS-LRTA* with the state-of-the-art LSS-LRTA* at solving real-time navigation problems in unknown environments. For fairness, we used comparable implementations that use the same underlying codebase. For example, both search algorithms use the same implementation for binary heaps as priority queues and break ties among cells with the same f-values in favor of cells with larger g-values.

We used six maps from deployed video games to carry out our experiments. The first three, from the game *Baldur's Gate*, are 512×512 cells each. The remaining three are 456×463, 939×718, and 656×1491 cells, and are taken from the game *Dragon Age*. All maps were retrieved from Nathan Sturtevant's repository.[1]

Each map in the repository has a set of associated problems. We estimated the hardness of the problems and selected the 300 hardest for each map. We estimate the hardness as the difference between the optimal cost to reach the goal from the initial state and the h-value for the initial state. All maps are regarded as undirected, eight-neighbor grids. Horizontal and vertical movements have cost 1, whereas diagonal movements have cost $\sqrt{2}$. We used the *octile distance* as heuristic.

[1] `http://www.movingai.com/`. For Baldur's Gate we used maps AR0011SR-512, AR0602SR-512, and AR0700SR-512. For Dragon Age, we used orz103d, orz702d, and orz900d.

	LSS-LRTA*			aLSS-LRTA*			%Cost
k	Avg. Cost	Time	Time/se	Avg. Cost	Time	Time/se	Impr.
1	1,330,352	1,222.2	0.001	981,828	888.9	0.001	26.2 ±1.8
4	424,693	785.6	0.003	330,933	609.4	0.003	22.1 ±1.6
7	285,568	657.4	0.004	228,333	527.4	0.004	20.0 ±1.6
10	215,545	557.9	0.006	182,361	474.2	0.006	15.4 ±1.8
13	171,338	482.8	0.007	147,743	417.9	0.008	13.8 ±1.9
16	139,467	422.4	0.009	125,512	382.5	0.009	10.0 ±1.6
19	121,376	392.9	0.011	109,596	357.2	0.011	9.7 ±1.5
22	106,256	366.0	0.012	95,806	332.5	0.012	9.8 ±1.3
25	95,783	350.0	0.014	87,019	320.8	0.014	9.1 ±1.3
28	86,328	333.1	0.015	78,603	306.2	0.016	8.9 ±1.3
31	78,582	319.5	0.017	72,395	297.4	0.017	7.9 ±1.4
34	73,834	314.3	0.018	67,666	292.2	0.019	8.4 ±1.4

Figure 3: Average results for the 6 game maps. For lookahead value k, we report the average solution cost (Avg. Cost), and two measures of efficiency: total runtime (Time), and average runtime per search episode (Time/se) in milliseconds. Results obtained over a Linux PC with a Pentium QuadCore 2.33 GHz CPU and 8 GB RAM.

We ran LSS-LRTA* and aLSS-LRTA* for 12 different lookahead values over each problem. Figure 3 shows average cost and time results. In addition, we show the percentage cost improvement of aLSS-LRTA* over LSS-LRTA*. The value after the "±" defines a 99% confidence interval for the cost improvement. This measure was obtained by carrying out a t-test for the cost difference for each individual problem. We observe aLSS-LRTA* consistently outperforms LSS-LRTA*, both in total search time and solution cost. The greatest improvements are obtained for lower values of the lookahead parameter (i.e., when there is more time pressure). On the other hand, since the additional overhead of aLSS-LRTA* (depression marking and move selection) is little: time per search episode is essentially the same.

The cost improvement on the 6 different maps we experimented with is shown in Figure 4. They present a rather similar behavior across different domains. This suggests that average values are representative of aLSS-LRTA*'s behavior.

Although aLSS-LRTA* is better suited for applications in which the environment is initially unknown, we also evaluated its performance in known environments (this implies the cost function c is initialized with the correct values). In this case we observed that aLSS-LRTA* outperforms LSS-LRTA* very consistently: improvements in this setting do not seem to depend on lookahead values. Indeed, cost-wise, aLSS-LRTA* outperforms LSS-LRTA* at least by 26% and at most by 40%; time-wise, aLSS-LRTA* improves on LSS-LRTA*'s total search time by least 30% and at most by 41%.

Relative Performance As shown by our experiments aLSS-LRTA* outperforms LSS-LRTA* on average. There are specific cases in which the situation is the opposite. Consider for example, path-finding scenarios that look like Figure 5. Because of the way ties are broken, aLSS-LRTA* always enters the shaded region, as the cells it leaves behind are marked as updated. When $height$ is lower than the length of *corridor* LSS-LRTA* also visits the shaded area. In those situations aLSS-LRTA* outperforms LSS-LRTA* because aLSS-LRTA* exits the shaded depressed region quickly. However, when $height$ is much larger than the length of the corridor,

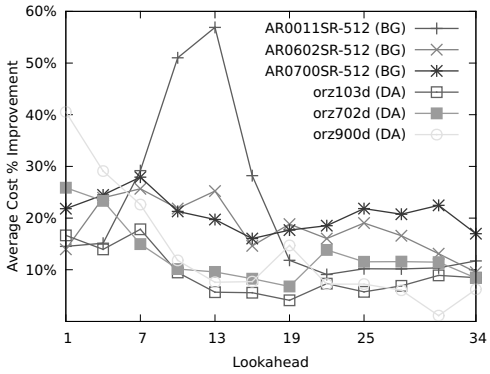

Figure 4: Cost improvement for different lookahead values on maps from Baldur's Gate (BG) and Dragon Age (DA).

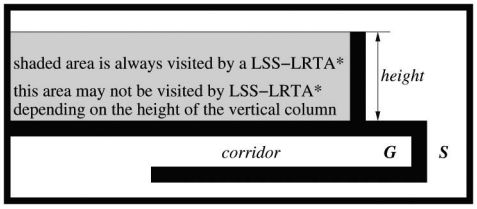

Figure 5: A situation in which the relative performance between LSS-LRTA* and aLSS-LRTA* changes depending on the value of $height$. S is the start state, and G is the goal. Ties are broken in favor of upper cells.

LSS-LRTA* does not enter the shaded region, producing better solutions. We did not observe such bad cases frequently in our experiments. Indeed, aLSS-LRTA* outperforms LSS-LRTA* in 75.6% of the problems we tried. We also computed the relative performance given by the ratio between the solution costs for each test case. On its 100 best cases, LSS-LRTA* obtains on average solutions 5.02 times cheaper than those of aLSS-LRTA*. On the other hand, aLSS-LRTA* obtains on average solutions that are 23.41 times cheaper than those of LSS-LRTA* on its 100 best cases.

6 Related Work

There are a number of real-time search algorithms that can be used in initially unknown environments. Besides LSS-LRTA*, RTAA* [Koenig and Likhachev, 2006] and LRTA*$_{\text{LS}}(k)$ [Hernández and Meseguer, 2007] are among the best-performing real-time search algorithms. LSS-LRTA* finds better-quality solutions than RTAA* for the same value of the lookahead parameter; however, LSS-LRTA* typically requires more time per search episode to obtain solutions of similar quality [Koenig and Likhachev, 2006]. LSS-LRTA*, on the other hand, is competitive with LRTA*$_{\text{LS}}(k)$ [Hernández and Meseguer, 2007]. None of the abovementioned algorithms utilize any mechanism comparable to depression avoidance. eLSS-LRTA* is a preliminary version of aLSS-LRTA* we presented in an extended abstract [Hernández and Baier, 2011]. It is outperformed by aLSS-LRTA* on average, as it usually becomes too focused on avoiding depressions.

Finally, less related to our work are algorithms that abide to real-time search constraints but that assume the environment is *known* in advance. Examples are D-LRTA* and kNN-LRTA* [Bulitko *et al.*, 2010], TBA* [Björnsson *et al.*, 2009].

7 Summary

We have presented a simple principle for guiding real-time search algorithms away from heuristic depressions. We integrated the principle into a state-of-the-art real-time search algorithm, producing aLSS-LRTA*, a novel heuristic real-time search algorithm. In aLSS-LRTA*, depressions are avoided by marking states in a depression using the Dijkstra algorithm and then selecting a state that is not in a depression as the next move, if possible. We showed aLSS-LRTA* is complete and preserves several of the properties of LSS-LRTA*. In our experiments, we observed that aLSS-LRTA* consistently outperforms LSS-LRTA*, most notably under time pressure.

Acknowledgements We thank the anonymous reviewers for their thoughtful insights. Carlos Hernández was partly funded a by Fondecyt project #11080063. Jorge Baier was funded by the VRI-38-2010 grant from Universidad Católica de Chile.

References

[Björnsson *et al.*, 2009] Y. Björnsson, V. Bulitko, and N. R. Sturtevant. TBA*: Time-bounded A*. In *IJCAI*, 431–436, 2009.

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

[Bulitko *et al.*, 2007] V. Bulitko, Y. Björnsson, M. Lustrek, J. Schaeffer, and S. Sigmundarson. Dynamic control in path-planning with real-time heuristic search. In *ICAPS*, 49–56, 2007.

[Bulitko *et al.*, 2010] V. Bulitko, Y. Björnsson, and R. Lawrence. Case-based subgoaling in real-time heuristic search for video game pathfinding. *Journal of Artificial Intelligence Research*, 38:268–300, 2010.

[Hart *et al.*, 1968] P. E. Hart, N. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimal cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 1968.

[Hernández and Meseguer, 2005] C. Hernández and P. Meseguer. LRTA*(k). In *IJCAI*, 1238–1243, 2005.

[Hernández and Meseguer, 2007] C. Hernández and P. Meseguer. Improving LRTA*(k). In *IJCAI*, 2312–2317, 2007.

[Hernández and Baier, 2011] C. Hernández and J. A. Baier. Escaping Heuristic Depressions in Real-Time Heuristic Search (Extended Abstract). In *AAMAS*, Taipei, Taiwan, May 2011.

[Ishida, 1992] T. Ishida. Moving target search with intelligence. In *AAAI*, 525–532, 1992.

[Koenig and Likhachev, 2006] S. Koenig and M. Likhachev. Real-time adaptive A*. In *AAMAS*, 281–288, 2006.

[Koenig and Sun, 2009] S. Koenig and X. Sun. Comparing real-time and incremental heuristic search for real-time situated agents. *Autonomous Agents and Multi-Agent Systems*, 18(3):313–341, 2009.

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

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

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