Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment

Real-Time Adaptive A* with Depression Avoidance

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

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

Abstract

RTAA* is probably the best-performing real-time heuristic search algorithm at path-finding tasks in which the environment is not known in advance or in which the environment is known and there is no time for pre-processing. As most real-time search algorithms do, RTAA* performs poorly in presence of heuristic depressions, which are bounded areas of the search space in which the heuristic is too low with respect to their border. Recently, it has been shown that LSS-LRTA*, a well-known real-time search algorithm, can be improved when search is actively guided away of depressions. In this paper we investigate whether or not RTAA* can be improved in the same manner. We propose aRTAA* and daRTAA*, two algorithms based on RTAA* that avoid heuristic depressions. Both algorithms outperform RTAA* on standard path-finding tasks, obtaining better-quality solutions when the same time deadline is imposed on the duration of the planning episode. We prove, in addition, that both algorithms have good theoretical properties.

Introduction

Several real-world applications require agents acting in complex environments to repeatedly act quickly in a possibly unknown environment. Such is the case of virtual agents that repeatedly perform pathfinding tasks in commercial games (e.g., *World of Warcraft*, *Baldur's Gate*, etc.). Indeed, it has been reported that game companies impose a time limit on the order of 1 millisecond for the path planning cycle that determines the move of every game character (Bulitko et al. 2010).

Real-time heuristic search (e.g. Korf 1990; Koenig 2001) is a standard approach used to solve these tasks. It is inspired on A* search (Hart, Nilsson, and Raphael 1968), but a mechanism is provided to bound the time taken to produce a movement. Real-time heuristic search algorithms repeatedly perform planning episodes whose computational time is bounded. Like standard A*, real-time heuristic algorithms use a heuristic function to guide action selection. Unlike A* search, the heuristic function is updated by the algorithm during execution. Such a process is usually referred to as the *learning* of the heuristic. The learning process guarantees that algorithms like LRTA* (Korf 1990) always finds a solution if any exists in finite, undirected search spaces.

Copyright © 2011, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Researchers (e.g. Ishida 1992) have observed that algorithms like LRTA* perform poorly when they enter *heuristic depressions*. Intuitively, a depression is a bounded connected component of the search space, D, in which the heuristic underestimates too much the cost to reach a solution in relation to the heuristic values of the states in the border of D. Most state-of-the-art real-time heuristic search algorithms (Bulitko et al. 2010; Koenig and Likhachev 2006; Hernández and Meseguer 2005; 2007; Koenig and Sun 2009) deal with this issue either by doing more *lookahead* while planning, or by increasing the amount of *learning*.

In a previous paper, we proposed aLSS-LRTA*, a variant of LSS-LRTA* (Koenig and Sun 2009) that *actively* guides search away of heuristic depressions; a principle called *depression avoidance* (Hernández and Baier 2011). aLSS-LRTA* typically outperforms LSS-LRTA*, especially when the lookahead phase is limited to explore few states.

Although relatively popular, LSS-LRTA* does not seem to be representative of the state-of-the-art in real-time heuristic search when time is taken into account; RTAA* (Koenig and Likhachev 2006), on the other hand, does seem to be. Indeed, Koenig and Likhachev (2006) show that RTAA* is *the* correct choice when the time per planning episode is bounded; we have verified those results, but omitted them for space. RTAA* is similar in spirit to LSS-LRTA*; its learning mechanism yields a less informed heuristic but it is faster and simpler to implement.

In this paper we report on whether or not depression avoidance can be incorporated into RTAA*, and propose two new real-time heuristic search algorithms. The first, aRTAA*, is a rather straightforward adaptation of aLSS-LRTA*'s implementation of depression avoidance into RTAA*. As aLSS-LRTA* does, it avoids moving to states that have been identified as belonging to a depression. We show aRTAA* outperforms RTAA* in path-finding tasks. The second algorithm, daRTAA*, is a finer implementation of depression avoidance that, like aRTAA*, will prefer to move to states that do not belong to a depression, but, unlike aRTAA*, when no such states are found, it prefers moving to states that seem closer to the border of the depression. We show daRTAA* outperforms aRTAA*. We prove both algorithms have nice theoretical properties: they maintain the consistency of the heuristic, and they terminate finding a

solution when such a solution exists.

The rest of the paper is organized as follows. We describe the basics of real-time heuristic search and argue for the significance of RTAA* in the next section. We follow describing the principle of depression avoidance and its implementation. Next, we present aRTAA* and daRTAA*, and analyze their properties. We finish with an experimental analysis and a discussion.

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 $(u, v) \in A$. We define $k(u, v)$ as a function that returns the cost of the minimum-cost path between states u and v. The successors of a state u are defined by $Succ(u) = \{v \,|\, (u, v) \in A\}$. 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 a 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, Nilsson, and Raphael 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.

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, Tovey, and Smirnov 2003)). The agent is capable of a limited form of sensing: only obstacles in the neighbor states can be detected. When obstacles are detected, the agent updates its map accordingly.

Most state-of-the-art real-time heuristic search algorithms can be described by the pseudo-code in Algorithm 1. The algorithm iteratively executes a lookahead-update-act cycle until the goal is reached. The lookahead phase (Line 5–7) determines the next state to move to, the update phase (Line 8) updates the heuristic, and the act phase (Line 9) moves the agent to its next position. The lookahead-update part of the cycle (Lines 5–8) is referred to as the *planning episode* throughout the paper.

The generic algorithm has three local variables: s stores the current position of the agent, $c(s, s')$ contains the cost of moving from state s to a successor s', and h is such that

Algorithm 1: A standard real-time heuristic search algorithm

```
   Input: A search problem P, a heuristic function h, a cost function c.
1  for each s ∈ S do
2      h_0(s) ← h(s)
3  s_current ← s_0
4  while s_current ∉ G do
5      LookAhead()
6      if Open = ∅ then return no-solution
7      s_next ← Extract-Best-State()
8      Update()
9      move the agent from s_current to s_next through the path
       identified by LookAhead. Stop if an action cost along the path
       is updated.
10     s_current ← current agent position
11     update action costs (if they have increased)
```

$h(s)$ contains the heuristic value for s. All three variables may change over time. In path-finding tasks, when the environment is initially unknown, the initial value of 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 $h(s)$, for every s, is given as a parameter.

In the lookahead phase (Line 5–7), the algorithm determines where to proceed next. The lookahead (Line 5) generates a search frontier of states reachable from s, which are stored in the variable $Open$. In RTA* and LRTA* (Korf 1990) the frontier corresponds to all states at a given depth d.[1] On the other hand, LSS-LRTA* (Koenig and Sun 2009), RTAA* (Koenig and Likhachev 2006), and other algorithms carry out an A* search that expands at most k states. The number k is referred to as the *lookahead parameter*, and the states generated during lookahead conform the so-called *local search space*. The variable $Open$ (cf. Line 6) contains the frontier of the local search space. Also, we assume that after executing an A* lookahead, the variable $Closed$ contains the states that were expanded by the algorithm. Finally, the next state to move to, s_{next}, is assigned in Line 7, and generally corresponds to the state s' in $Open$ that minimizes the sum $k(s_{current}, s')+h(s')$, where $k(s_{current}, s')$ is cost of the optimal path from $s_{current}$ to s'.

When an A* lookahead is used with consistent heuristics, such a state is the one with minimum f-value in $Open$ (see Algorithm 2).

Algorithm 2: Selection of the Best State used by LSS-LRTA*, RTAA*, and others

```
   Input: A search problem P, a heuristic function h, a cost function c.
1  procedure Extract-Best-State()
2      return argmin_{s'∈Open} g(s') + h(s')
```

Using the heuristic of all or some of the states in the frontier of the local search space ($Open$), the algorithm updates the heuristic value of states in the local search space (Line 8). Intuitively, after the lookahead is carried out, information is gained regarding the heuristic values of states in the frontier of the local search space. This information is used to up-

[1] These algorithms are usually described assuming $d = 1$.

date the h-value of states in the local search space in such a way that they are consistent with the h-values of the frontier. As before, different algorithms implement different mechanisms to update the heuristics. In what follows, we focus on LSS-LRTA* and RTAA*, since these are the most relevant algorithms to this paper.

LSS-LRTA* updates the values of each state s in the local search space in such a way that $h(s)$ is assigned the *maximum* possible value that guarantees consistency with the states in $Open$. It does so by implementing the `Update` procedure as a version of Dijkstra's algorithm (see Koenig and Sun's paper for details). Since the value of h is raised to the maximum, the update mechanism of LSS-LRTA* makes h as informed as it can get while maintaining consistency.

Algorithm 3: RTAA*'s Update Procedure

```
   Input: A search problem P, a heuristic function h, a cost function c.
1  procedure Update ()
2      f* ← min_{s∈Open} g(s) + h(s)
3      for each s ∈ Closed do
4          h(s) ← f* − g(s)
```

RTAA*, on the other hand, uses a simpler update mechanism. It updates the heuristic value of states in the interior of the local search space (i.e. those stored in A*'s variable $Closed$) using the f-value of the best state in $Open$. The procedure is shown in Algorithm 3. The heuristic values that RTAA* learns may be less informed than those of LSS-LRTA*. The following two propositions establish this relation formally, and, to our knowledge, are not stated explicitly in the literature.

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

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

where h_n denotes the value of the h variable at the start of iteration n and after the update of iteration $n-1$. $k_n(s, s_b)$ denotes the cost of the cheapest path from s to s_b, with respect to the c variable at iteration n and that only traverses states in $Closed$ before ending in s_b. For RTAA*, the situation is slightly different.

Proposition 2 *Right after the call to A* in the n-th iteration of RTAA*, let s^* be the state with lowest f-value in Open, and let s be a state in $Closed$. Then,*

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

However, if h_n is consistent and s is in the path found by A* from $s_{current}$ to s^*, then

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

Proposition 2 implies that, when using consistent heuristics, RTAA*'s update yields possibly less informed h-values than those of LSS-LRTA*. However, at least for some of the states in the local search space, the final h-values are equal to those of LSS-LRTA*, and hence they are as informed as they can be.

An advantage of RTAA*'s update scheme over that of LSS-LRTA* is that the former runs faster. When given a time deadline of T units of time per planning episode, RTAA* will find *better* solutions than LSS-LRTA* (Koenig and Likhachev 2006). We carried out an independent empirical evaluation over 12 game maps in which we confirmed that observation. For example, given a deadline of 0.005 milliseconds for the planning episode, RTAA* finds solutions on average 11.6% cheaper than those found by LSS-LRTA*. For a deadline of 0.02 milliseconds, RTAA* finds solutions on average 41.5% cheaper than those found by LSS-LRTA*. We concluded that RTAA* seems superior to LSS-LRTA* when time per planning episode is restricted.

Depression Avoidance

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.

Ishida (1992) was perhaps the first to analyze the behavior of real-time heuristic search algorithms in presence of such regions. For Ishida, a depression is a maximal connected component of states that defines a *local minimum* of h. Real-time search algorithms like LRTA* become trapped in these regions, precisely because movements are guided by the heuristic. As such, once an agent enters a depression, the only way to leave it is by raising the heuristic values of the states in the depression high enough as to make the depression disappear. Algorithms capable of performing more learning than LRTA*, such as LSS-LRTA* or RTAA*, also perform poorly in these regions because their movements are also only guided by the value of the heuristic.

In previous work, we proposed a definition for *cost-sensitive* heuristic depressions that is an alternative to Ishida's and takes costs into account (Hernández and Baier 2011). Intuitively, a state s is in a cost-sensitive heuristic depression if its heuristic value is not a realistic cost estimate with respect to the heuristic value of every state in the border, considering the cost of reaching such a state from s. Formally,

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 and traverses states only in D before ending in s'.*

aLSS-LRTA*

aLSS-LRTA* (Hernández and Baier 2011) is a modification of LSS-LRTA* that implements *depression avoidance*: a principle that dictates that search should be guided away from states identified as being in a heuristic depression. It is a simple but effective modification of LSS-LRTA*. When deciding the next move aLSS-LRTA* prefers states that are not yet marked as belonging to a depression. Based on the

observation that Dijkstra's algorithm will update the heuristic of a state if it's in a cost-sensitive depression of the local search space, a state is marked as being part of a depression when its heuristic value is updated. To select the next move, the algorithm chooses the best state in $Open$ that has *not* been marked as in a depression. If such a state does not exist the algorithm selects the best state in $Open$, just like LSS-LRTA* would do.

The selection of the state to move to is implemented by the function in Algorithm 4, where s.updated denotes whether or not the heuristic of s has been updated (i.e., whether or not a state is marked). The update procedure is modified appropriately to set this flag. Despite the fact

Algorithm 4: Selection of the Next State used by aLSS-LRTA* and aRTAA*

```
1 function Extract-Best-State ()
2     if Open contains an s such that s.updated = false then
3         s ← argmin_{s'∈Open∧s'.updated=false} g(s') + h(s')
4     else
5         s ← argmin_{s'∈Open} g(s') + h(s')
6     return s ;
```

that aLSS-LRTA* does not necessarily move to the best state in $Open$, it is guaranteed to find a solution in finite undirected search spaces if such a solution exists. In path-finding benchmarks, aLSS-LRTA* improves the solution cost over LSS-LRTA* by about 20% for small lookahead values and by about 8.4% for high values of the lookahead parameter.

RTAA* with Depression Avoidance

In this section we propose two variants of the state-of-the-art RTAA* algorithm that implement depression avoidance. The first, aRTAA*, is based on aLSS-LRTA*. The second, daRTAA*, is a more fine-grained adaptation of depression avoidance which prefers moving to states that seem closer to the border of a depression.

aRTAA*

aRTAA* is a straightforward port of aLSS-LRTA*'s implementation of depression avoidance into RTAA*. RTAA* is modified as follows. First, its update procedure is replaced by Algorithm 5, which implements the same update rule of RTAA* but, like aLSS-LRTA*, marks states that have been updated. Second, RTAA*'s procedure to select the next state is replaced by that of aLSS-LRTA* (Algorithm 4). As a result aRTAA* is a version of RTAA* that avoids depressions using the same mechanism that aLSS-LRTA* utilizes.

Properties of aRTAA* aRTAA* inherits a number of RTAA*'s properties. Since the update rule is not changed, we can use the same proofs by Koenig and Likhachev (2006) to show that h is non-decreasing over time, and that h remains consistent if so it is initially. Other properties specific to aRTAA* can also be shown.

Theorem 1 *Let s be a state such that s.updated switches from false to true between iterations n and $n+1$ in an execution of aRTAA* initialized with a consistent heuristic h. Then s is in a cost-sensitive heuristic depression of h_n.*

Algorithm 5: aRTAA*'s Update Procedure

```
1 procedure Update ()
2     if first run then
3         for each s ∈ S do s.updated ← false
4     f* ← min_{s∈Open} g(s) + h(s);
5     for each s ∈ Closed do
6         h(s) ← f* − g(s);
7         if h(s) > h_0(s) then s.updated ← true
```

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 aRTAA*, used with h, will find a solution for P.*

daRTAA*

daRTAA* is based on aRTAA*, but differs from it in the strategy used to select the next state to move to. To illustrate the differences, consider a situation when there is no state s in the frontier such that s.updated is false. In this case, aRTAA* behaves exactly as RTAA* does. This seems rather extreme, since intuitively we would still like the movement to be guided away of the depression. In such situations, daRTAA* will actually attempt to escape the depression by choosing the state with best f-value among the states whose heuristic has *changed the least*. The intuition behind this behavior is as follows: assume $\Delta(s)$ is the difference between the actual cost to reach a solution from a state s and the initial heuristic value of state s. Then if s_1 is a state close to the border of a depression D and s_2 is a state farther away from the border and "deep" in the interior of D, then $\Delta(s_2) \geq \Delta(s_1)$, because the heuristic of s_2 is more imprecise than that of s_1. At execution time, h is an estimate of the actual cost to reach a solution. As such, in summary, daRTAA* always moves to a state believed not to be in a depression, but if no such state exists it moves to states that are regarded as closer to the border of the depression.

daRTAA* is implemented like RTAA* but the procedure to select the next state to move to is given by Algorithm 6.

Algorithm 6: daRTAA*'s Selection of the Next State

```
1 function Extract-Best-State ()
2     Δ_min ← ∞ ;
3     while Open ≠ ∅ and Δ_min ≠ 0 do
4         Remove state s_b with smallest f-value from Open;
5         if h(s_b) − h_0(s_b) < Δ_min then
6             s ← s_b ;
7             Δ_min ← h(s_b) − h_0(s_b);
8     return s
```

Properties of daRTAA* Since only the mechanism for selecting the next move is modified, daRTAA* inherits directly

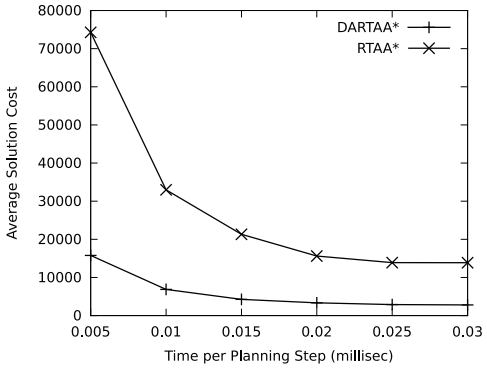

Figure 1: A comparison of best solution costs (averaged) obtained by daRTAA* and RTAA* given a fixed time deadline.

most of the properties of RTAA*. In particular, h is non-decreasing over time, and consistency is maintained. We can also prove termination.

Theorem 3 *Let P be an undirected finite real-time search problem such that a solution exists. Let h be a consistent heuristic for P. Then daRTAA*, used with h, will find a solution for P.*

Experimental Evaluation

We compared RTAA*, aRTAA* and daRTAA* 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 12 maps from deployed video games to carry out the experiments. The first 6 are taken from the game *Dragon Age*, and the rest are taken from the game *StarCraft*. The maps were retrieved from Nathan Sturtevant's repositories.[2]

We average our results over 6,000 test cases (500 test cases for each game map). Each test case is random. Maps are undirected, eight-neighbor grids, and the agent is capable of observing obstacles in neighbor cells. Horizontal and vertical movements have cost 1, whereas diagonal movements have cost $\sqrt{2}$. We used the *octile distance* as heuristic.

Figure 2 shows average results for RTAA*, aRTAA* and daRTAA* for 17 different lookahead values. For reference, we include *Repeated A**, a search algorithm for unknown environments that uses *unbounded* A* in the lookahead phase, and that does not update the heuristic values. We observe that in terms of solution cost, for all lookahead values, aRTAA* consistently outperforms RTAA*; moreover, daRTAA* outperforms aRTAA*. Nevertheless, for any lookahead value, aRTAA* spends more time per planning episode than RTAA* does; this can be explained by the extra condition aRTAA* has to check for each state that is updated. daRTAA*, on the other hand, spends more time than RTAA* per planning episode. This increase is due to daRTAA*'s selection of the next state to move to is less efficient. In RTAA* this selection is quick, since it only involves extracting the best state in $Open$, which can be done in constant time with binary heaps. daRTAA*, on the other hand, may extract several states from the open list per planning episode. Thus we observe a higher number of heap percolations. The worst-case time complexity of daRTAA*'s selection procedure is $O(n \log n)$, where n is the size of $Open$.

The experimental results show that daRTAA*'s more refined mechanism for escaping depressions is better than that of aRTAA*. For small values for the lookahead parameter, daRTAA* obtains better solutions than aRTAA* used with a much larger lookahead. For example, with a lookahead parameter equal to 1, daRTAA* obtains better solutions than aRTAA* with lookahead parameter equal to 19, requiring, on average, 10 times less time per planning episode.

daRTAA* substantially improves RTAA*, which is probably the best real-time heuristic search algorithm known to date. daRTAA* needs only a lookahead parameter of 25 to obtain solutions better than RTAA* with lookahead parameter of 97. With those values, daRTAA* requires about 2.6 times less time per planning episode than RTAA*.

Figure 1 shows a plot of the average best solution quality obtained by daRTAA* and RTAA* given fixed time deadlines per planning episode. For both algorithms, the solution quality improves as more time is given per planning episode. For every time deadline plotted, daRTAA* obtains much better solutions, especially when the time deadline is small.

Experimentally, daRTAA* is clearly superior to RTAA*. Of the 102,000 runs, daRTAA* obtains a better solution quality than RTAA* on 65.8% of the cases, they tie on 24.8% of the cases, and RTAA* obtains a better-quality solution in only 9.4% of the cases. In addition, we computed the relative performance of the algorithms, which is given by the ratio between the solution costs for each solved problem. In the 10,000 cases in which the ratio is most favorable to RTAA*, the solutions obtained by RTAA* are 1.43 times cheaper on average than those obtained by daRTAA*. On the other hand, in the 10,000 cases in which the ratio is most favorable to daRTAA* over RTAA*, the former algorithm obtains solutions that are 2.79 times cheaper on average.

If one analyzes the performance on the hardest test cases for each algorithm, i.e. those for which the highest solution costs were obtained, we conclude again in favor of daRTAA*. Indeed, in the 10,000 hardest test cases for RTAA*, solutions obtained by RTAA* are on average 8.37 times more expensive than those of daRTAA*. On the other hand, in the 10,000 hardest test cases for daRTAA*, solutions obtained by daRTAA* are 5.57 times cheaper than those obtained by RTAA*.

Finally, we observe all real-time algorithms generate worse solutions than Repeated A*. However, they do so in significantly less time. For example, daRTAA* with lookahead 97 obtains a solution 5.1 times worse than A*, but uses an order of magnitude more time per planning episode.

[2] `http://www.movingai.com/` and `http://hog2.googlecode.com/svn/trunk/maps/`. For Dragon Age we used the maps brc202d, orz103d, orz702d, ost000a, ost000t and ost100d of size 481 x 530, 456 x 463, 939 x 718, 969x 487, 971 x 487 and 1025 x 1024 cells respectively. For StarCraft, we used the maps Enigma, FadingRealm, JungleSiege, Ramparts, TwistedFate and WheelofWar of size 768 x 768, 384 x 512, 768 x 768, 512 x 512, 384 x 384 and 768 x 768 cells respectively.

k	Solution Cost	# Planning Episodes	Time per Episode	Time	Percolations per episode
			RTAA*		
1	553,152	510,579	0.0004	183.4	5.7
7	197,781	107,321	0.0016	174.7	39.9
13	109,587	43,835	0.0029	127.8	86.0
19	75,239	24,932	0.0042	104.6	139.9
25	56,989	16,606	0.0055	90.9	198.3
31	46,201	12,191	0.0068	82.5	259.1
37	38,839	9,457	0.0081	76.3	323.9
43	33,675	7,679	0.0094	71.9	389.8
49	29,658	6,403	0.0107	68.4	457.6
55	26,424	5,445	0.0120	65.4	525.8
61	23,971	4,745	0.0133	63.3	593.3
67	21,811	4,173	0.0147	61.2	659.8
73	20,264	3,761	0.0160	60.2	729.7
79	18,857	3,410	0.0173	59.1	794.3
85	17,524	3,099	0.0187	58.0	861.6
91	16,523	2,865	0.0201	57.5	926.3
97	15,422	2,632	0.0214	56.4	992.4
			aRTAA*		
1	432,806	399,603	0.0005	197.0	9.0
7	146,724	80,528	0.0021	170.0	61.3
13	82,311	33,795	0.0036	121.7	118.4
19	57,232	19,661	0.0050	99.2	181.6
25	45,336	13,660	0.0065	88.2	244.7
31	38,000	10,347	0.0079	81.5	310.1
37	31,290	7,890	0.0093	73.1	380.5
43	27,239	6,450	0.0107	68.8	450.1
49	24,143	5,427	0.0121	65.4	521.1
55	21,908	4,712	0.0135	63.4	590.9
61	19,692	4,080	0.0148	60.5	661.0
67	18,450	3,696	0.0162	59.8	729.8
73	17,118	3,338	0.0176	58.7	800.5
79	15,933	3,031	0.0190	57.6	870.9
85	14,975	2,791	0.0204	56.9	938.3
91	14,171	2,590	0.0217	56.3	1,004.0
97	13,099	2,362	0.0231	54.6	1,071.1
			daRTAA*		
1	50,906	47,836	0.0005	19.5	9.1
7	28,932	22,564	0.0023	51.3	90.8
13	22,323	15,704	0.0043	67.6	179.9
19	16,460	10,592	0.0063	66.5	266.5
25	14,116	8,393	0.0082	69.1	351.9
31	12,657	6,924	0.0100	69.5	434.4
37	10,619	5,433	0.0118	64.0	518.2
43	10,771	5,192	0.0134	69.7	603.7
49	9,576	4,436	0.0151	66.9	680.8
55	9,059	4,030	0.0166	66.9	761.0
61	8,195	3,500	0.0180	63.1	835.7
67	7,446	3,047	0.0194	59.0	909.0
73	7,073	2,809	0.0208	58.4	985.8
79	6,628	2,533	0.0220	55.9	1,057.7
85	6,606	2,468	0.0235	58.0	1,128.3
91	6,346	2,297	0.0246	56.6	1,194.8
97	6,397	2,279	0.0263	59.8	1,265.0
			Repeated A*		
∞	1,265	571	0.57	328	

Figure 2: The table presents average solution cost, number of planning episodes, time per planning episode in milliseconds, total search time in milliseconds, and number of heap percolations per planning episode for 6,000 path-planning tasks and 17 lookahead values (k). We performed our experiments on a Linux PC with a Pentium QuadCore 2.33 GHz CPU and 8 GB RAM.

Discussion

Although daRTAA* and aRTAA* clearly outperform RTAA* in our experiments, we believe it is possible to contrive families of increasingly difficult path-finding tasks in which daRTAA* or aRTAA* will find solutions arbitrarily worse than those found by RTAA*. Those situations exist for aLSS-LRTA* (Hernández and Baier 2011). Nevertheless, it is not clear that the existence of these families of "worst-case situations" are a real concern from a practical point of view. On the one hand, we did not spot a significant number of these cases in our experimental evaluation. On the other hand, it is easy to come up with families of problems in which daRTAA* and aRTAA* perform arbitrarily *better* than RTAA*.

Summary and Conclusions

We have proposed aRTAA* and daRTAA*, two real-time heuristic search algorithms that implement depression avoidance on top of the state-of-the-art RTAA*. aRTAA* is a straightforward adaptation of the implementation of aLSS-LRTA*. daRTAA* is a more fine-grained implementation of depression avoidance that, when trapped in a depression, prefers to move to states that seem to be closer to the border of such a depression. We showed both algorithms outperform RTAA*, and that daRTAA* is the best of the three. We believe the particular implementation of depression avoidance that we devised for daRTAA* holds promise, since it could be easily incorporated in other search algorithms in order to find better-quality solutions when time is bounded.

References

Bulitko, V.; Björnsson, Y.; Sturtevant, N.; and Lawrence, R. 2010. *Real-time Heuristic Search for Game Pathfinding*. Applied Research in Artificial Intelligence for Computer Games. Springer.

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

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

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

Hernández, C., and Baier, J. A. 2011. Real-time heuristic search with depression avoidance. In *IJCAI*.

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

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

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

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

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

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