Weighted Real-Time Heuristic Search

Nicolás Rivera
Depto. de Ciencia de la Computación,
Pontificia Universidad Católica de Chile
nnrivera@uc.cl

Jorge A. Baier
Depto. de Ciencia de la Computación,
Pontificia Universidad Católica de Chile
jabaier@ing.puc.cl

Carlos Hernández
Depto. de Ingeniería Informática,
Universidad Católica de la Ssma. Concepción
chernan@ucsc.cl

ABSTRACT

Multiplying the heuristic function by a weight greater than one is a well-known technique in Heuristic Search. When applied to A* with an admissible heuristic it yields substantial runtime savings, at the expense of sacrificing solution optimality. Only a few works have studied the applicability of this technique to Real-Time Heuristic Search (RTHS), a search approach that builds upon Heuristic Search. In this paper we present two novel approaches to using weights in RTHS. The first one is a variant of a previous approach by Shimbo and Ishida. It incorporates weights to the lookahead search phase of the RTHS algorithm. The second one incorporates the weight to the edges of the search graph during the *learning* phase. Both techniques are applicable to a wide class of RTHS algorithms. Here we implement them within LSS-LRTA* and LRTA*-LS, obtaining a family of new algorithms. We evaluate them in path-planning benchmarks and show the second technique yields improvements of up to one order-of-magnitude both in solution cost and total search time. The first technique, on the other hand, yields poor results. Furthermore, we prove that RTHS algorithms that can appropriately use our second technique terminate finding a solution if one exists.

Categories and Subject Descriptors

I.2.8 [**Problem Solving, Control Methods, and Search**]: Graph and Tree Search Strategies, Heuristic Methods

General Terms

Algorithms, Experimentation

Keywords

A*, Weighted A*, Learning Real-Time A*, Dijkstra's Algorithm, Real-Time Heuristic Search

1. INTRODUCTION

Weighted A* [12] is a well-known search algorithm for single-agent search problems. It introduced the technique of multiplying the heuristic function by a weight. Based on A* [4], it uses an evaluation function $f(s) = g(s) + wh(s)$ to rank a state s in the search frontier, where $g(s)$ represents the cost incurred to reach s, $h(s)$, is a (heuristic) estimate of the cost to reach a solution from s, and the *weight* w is a real value greater or equal to one. It can find a solution substantially faster than A* as w is increased. However, the cost of such a solution is often non-optimal, and usually increases as w is increases. If the heuristic h is admissible, it can be at most a factor w away from optimal.

Weighting the heuristic is a simple but powerful technique that is widely used in state-of-the-art Heuristic Search algorithms. For example, ARA* [11], an algorithm used in outdoor rover applications, and RWA* [13], the search algorithm underlying LAMA 2011 – among the best-performing satisficing automated planners – rely on this technique to obtain superior performance. Nevertheless, in Real-Time Heuristic Search (RTHS) [10], an approach to solving search problems under tight time constraints, with applications ranging from video games to highly dynamic robotics, existing approaches that use weights have not shown improved performance when the objective is to find a single solution. This is surprising since RTHS is based on Heuristic Search.

Shimbo and Ishida [14] introduced *Weighted LRTA** an application of weights to an RTHS algorithm. Their algorithm multiplies the heuristic function by a weight w at the outset and solves the resulting problem with an RTHS algorithm. In their experiments, although they seem to improve convergence performance, they show poor-to-modest performance at finding a single solution. The LRTS algorithm [3] uses lower-than-one weights applied to the costs of the graph. In terms of movement, this has the effect of indirectly giving more importance to the heuristic h, when w is decreased. However, in terms of learning, this kind of update yields a weaker heuristic than a standard update. LRTS's performance is consistent with the results obtained by Shimbo and Ishida: the solution cost worsens as the weight is decreased [1].

In this paper we propose two new approaches to incorporating weights in RTHS. Both are applicable to a wide range of RTHS algorithms. The first approach, *weighted lookahead*, uses Weighted A* in the lookahead step of the RTHS algorithm. It can be seen as a variant of Shimbo and Ishida's approach but, as we see later, outperforms it in practice. The second approach, *weighted update*, incorporates w in h by using a different learning rule in which, the higher the w, the higher the amount by which the heuristic may be increased in every update (learning) step. Both approaches are very simple to implement in standard RTHS algorithms.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013),* Ito, Jonker, Gini, and Shehory (eds.), May 6–10, 2013, Saint Paul, Minnesota, USA.

We implement both approaches on top of the state-of-the-art algorithms LSS-LRTA* [8] and LRTA*-LS [5], producing four new algorithms. We evaluate the algorithms over standard video-game path-finding tasks. We show that the weighed lookahead approach, like Shimbo and Ishida's approach, yields both worse solutions and worse running times as w increases. On the other hand, we show that weighted update does yield benefits in *both* solution quality *and* runtime. Improvements are up to one order of magnitude when the algorithm parameter (a measure of the search effort per iteration) is small. The fact that improvements are observed in both solution quality and search time is rather interesting, since, in Heuristic Search algorithms, weights usually increase solution cost. Furthermore, we evaluate our best-performing technique theoretically, and show that algorithms that use weighted update, under certain conditions, will always find a solution if one exists, along with other relevant properties.

The remainder of the paper is organized as follows. The next section introduces background on RTHS, LSS-LRTA*, and LRTA*-LS. Then we describe the two proposed approaches and how they can be implemented within LSS-LRTA* and LRTA*-LS. We then evaluate the weighted update approach theoretically. The next section evaluates our algorithms empirically. Then we discuss relevant aspects about the performance of the algorithms we analyze. The paper finishes with a summary and conclusions.

2. BACKGROUND

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. We assume that S is finite, that A does not contain elements of form (s, s), and that (S, A) is a strongly connected graph. In addition, we have a cost function $c : A \to \mathbb{R}^+$ which associates a cost with each of the available actions. For all $s \in S$ we define $Succ(s) = \{t \in S : (s, t) \in A\}$. Finally $G \subseteq S$ is a set of goal states. The distance function $d : S \times S \to \mathbb{R}$ is such that $d(s, t)$ – the distance between s and t – denotes the cost of a shortest-path between s and t. Given a subset T of S we define the frontier of T as $\partial T = \{s \in S \setminus T : \exists t \in T$ such that $(t, s) \in A\}$. Furthermore, we define d_T as the cost of a shortest-path of the form $t_0 \cdots t_n$, where $t_0, \ldots, t_{n-1} \in T$ and $t_n \in T \cup \partial T$.

A heuristic function $h : S \to [0, \infty)$ associates to each state s an approximation $h(s)$ of the cost of a path from s to a goal state. We denote by $h^*(s)$ the distance from s to a goal state. A heuristic h is *consistent* if and only if (1) $h(g) = 0$ for every $g \in G$, and (2) for any $(s, t) \in A$ it holds that $h(s) \leq c(s, t) + h(t)$. If h is consistent then $h(s) \leq d(s, t) + h(t)$ for all $s, t \in S$. Furthermore, if h is consistent it is easy to prove that it is also *admissible*; i.e., $h(s)$ underestimates $h^*(s)$. We say that a state t *justifies* the h-value of state s if $h(s) = c(s, t) + h(t)$.

We assume familiarity with the A* algorithm [4]: $g(s)$ 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, g-value, and h-value of s refer to $f(s)$, $g(s)$, and $h(s)$ respectively. The variable *Closed* contains a set of nodes that have been expanded, and *Open* contains the set of nodes generated by the algorithm that are not in *Closed*. We also use the fact that, right after A* expands a node, $Open = \partial Closed$, which is simple to prove by induction on the number of A* iterations.

2.1 Real-Time Heuristic Search

In RTHS, the objective is to move an agent from an initial state to a goal state. Between each movement, the computation carried out by the algorithm should be bounded by a constant. An example situation is path-finding in a priori unknown grid-like environments. In this situation, we assume the agent knows the dimensions of the grid but not the location of the obstacles before the search is started.

Most RTHS algorithms iterate three steps until they find the solution. In the *lookahead* step, the agent runs a heuristic search algorithm to search for a next move. In the *movement* step, the agents moves to a different position. If the environment is initially unknown, in the movement step the agent also updates its knowledge about the search graph. Finally, in the *update* step, the agent will update the h-value of some of the states in the search space. The update step is usually necessary to guarantee that the algorithm will find a solution. The performance of RTHS algorithms is sensitive to the way in which the heuristic is updated (see e.g., [7]). Finally, we note that the order in which the three steps are carried out depends on the particular algorithm.

Our experimental evaluation focuses on path-finding in grid-like, a priori unknown terrain. RTHS algorithms in a priori unknown environments assume that prior to search the agent knows the structure of the graph but does not know the cost function c. While moving through the environment, however, the agent can observe that some arcs in the graph have a cost that is *greater* than the cost it currently knows. In our experiments we undertake the *free-space assumption* [16, 9], a standard assumption about the initial knowledge of the agent, whereby the terrain is initially assumed obstacle-free. The agent, on the other hand, can observe obstacles in the immediate neighborhood of the current cell. When obstacles are detected, the agent updates its cost function accordingly, by setting the cost of reaching a previously unknown obstacle cell to infinity.

Below we present two general techniques that can be applied to a range of RTHS algorithms. To illustrate their applicability, we implement the techniques within two state-of-the-art RTHS algorithms: LSS-LRTA* [8] and LRTA*-LS [5]. Below, we describe them in more detail.

LSS-LRTA* (Algorithm 1) is a generalization of the well-known LRTA* algorithm [10]. Its lookahead procedure invokes a bounded A* algorithm which expands at most k nodes. At the end of A* the states in *Closed* are usually referred to as the *local search space*. After lookahead, the h-values of the states in the interior of the local search space are updated. The update formula (Eq. 1; Alg. 1) is such that the resulting h-value of s is the maximum possible value that still preserves consistency [8]. Finally, in the movement step, the algorithm moves the agent as far as possible towards the best state in *Open*, observing the environment, and updating the cost function.

LRTA*-LS (Algorithm 2) is an RTHS algorithm that differs from LSS-LRTA* mainly in how it builds the region of states for the update. In each iteration, LRTA*-LS builds a *learning space*, denoted by I in Alg. 2. It does so by running a breadth-first search from the current state, which will add a state s to I if $h(s)$ is not justified by any of its successor states outside of I. Just like LSS-LRTA*, LRTA*-LS updates the h-values of states in I to the maximum possible value that preserves consistency (Eq. 1; Alg. 1). Finally, in the movement step, it moves the agent to the best neighbor.

Algorithm 1: LSS-LRTA*

```
Input: A search problem $P$, a heuristic function $h$, and
       a lookahead parameter $k$
1 while the agent has not reached a goal state do
2     Lookahead: Perform an A* search rooted at the current state. Stop as soon as $k$ nodes have been
      expanded and added to Closed. Furthermore, if just before extracting a node from Open a goal state $g$
      has minimum $f$-value in Open, stop A* before extracting $g$ from Open.
3     Update: Update the $h$-values of each state $s$ in Closed such that
          $h(s) := \min_{t \in Open} d_{Closed}(s,t) + h(t)$.        (1)
4     Movement: Let best be the state with lowest $f$-value in Open. Move towards best along the path
      identified by A*. While moving, observe the environment and update the cost function when
      new obstacles are found. Stop as soon as best is reached or when an obstacle blocks a state in the
      path to best.
```

Algorithm 2: LRTA*-LS

```
Input: A search problem $P$, a heuristic function $h$, and
       a parameter $k$
1 while the agent has not reached a goal state do
2     Update: Build a set of states $I$ as follows.
      Initialize a queue $Q$ as containing the current state.
      Let $I := \emptyset$. Now, until $|I| = k$ or $Q$ is empty, pop an element $s$ from $Q$, and if
      $h(s) < c(s,t) + h(t)$ for every $t \in Succ(s) \setminus I$, then (1) add $s$ to $I$, and (2)
      push to $Q$ all successors of $s$ not in $I$. Finally, update the $h$-values of every state $s \in I$ such that
          $h(s) := \min_{t \in \partial I} d_I(s,t) + h(t)$.        (2)
3     Lookahead: Let the current state be $s$. Set next to $\arg\min_{t \in Succ(s)} c(s,t) + h(t)$.
4     Movement: Move the agent to next, observe the environment and update the costs of the search
      graph when new obstacles are found.
```

Algorithm 3: Modified Dijkstra's Algorithm

```
Input: A region of states $I$
Effect: If $s \in I$, $h(s)$ is set to $\min_{t \in \partial I} d_I(s,t) + h(t)$
1 $R := I \cup \partial I$
2 for each $s \in I$ do $h(s) := \infty$
3 while $R \neq \emptyset$ do
4     Let $t$ be the state with lowest $h$-value in $R$
5     for each $s \in I$ such that $t \in Succ(s)$ do
6         if $h(s) > c(s,t) + h(t)$ then
7             $h(s) := c(s,t) + h(t)$
8     remove $t$ from $R$
```

Note that both LSS-LRTA* and LRTA*-LS update equations are exactly the same since $\partial Closed = Open$. As such, both use a modified version of Dijkstra's algorithm – shown in Algorithm 3 – to update the heuristic. The algorithm receives a region of nodes I as input and recomputes the h-values of states in I by interpreting the h function as the cost of a shortest path between the frontier ∂I and I. As a result, it sets h-values of states in I according to Equation 2 in Algorithm 2. References [8] and [6] provide details and proofs of correctness.

In summary, for LSS-LRTA* the local search space and the learning space are the same and are given by the states in *Closed* after k iterations of an A* run starting from the current state. For LRTA*-LS the learning space is the set constructed in Line 2 of Algorithm 2 and the local search space is just the current state.

3. WEIGHTED RTHS

Now we describe two approaches to incorporating weights into the heuristic function of RTHS algorithms. As we show later, the weighted lookahead approach does not yield good results in our benchmark problems. We think however that it deserves to be discussed here because it is the obvious way in which the idea of Weighted A* can be adapted to RTHS, and thus relevant conclusions can be obtained by analyzing it theoretically and empirically.

3.1 Weighted Lookahead

The *weighted lookahead* approach consists of using Weighted A* in the lookahead phase of the RTHS algorithm. It is directly applicable to any RTHS algorithm that uses A* in its lookahead phase, but may also be applied to algorithms that use different lookahead procedures.

In this paper we consider incorporating it into LSS-LRTA* and LRTA*-LS, and we call the resulting algorithms LSS-LRTwA* and LRTwA*-LS. Straightforwardly, LSS-LRTwA* differs from LSS-LRTA* in that Weighted A* instead of A* is called in Line 2 of Alg. 1, with the stop condition left intact.

For LRTwA*-LS, we interpret the selection of the next state to move to in Line 3 of Alg. 2 as an A* search that expands only the current state and selects the best state in its neighborhood. We thus modify it to set *next* to the state that minimizes $c(s,t) + wh(t)$, where s is the current state and t ranges over its successors.

3.2 Weighted Update

A possible reason that explains why Weighted A* finds solutions more quickly than regular A* is that in multiplying the heuristic by a factor $w \geq 1$, the heuristic becomes more *accurate*, in a significant portion of the search space. This is sensible since in many search problems heuristics sometimes grossly underestimate the true cost to reach a solution. RTHS is no different from Heuristic Search in this respect, as usually inaccurate heuristics can be used. Thus by multiplying the heuristic by a factor greater than 1 one would expect the heuristic to become more accurate in many parts of the search space. Unfortunately, as we show later, incorporating weights in the lookahead, as done by the previous approach, does not work well in practice. Here we consider incorporating weights in an alternative way.

The main idea underlying *weighted update* is to make h increase by a factor of w using the update procedure of the

RTHS algorithm. To accomplish this, in the update phase we run the standard update algorithm (i.e., Dijkstra) but in a *modified* region I, in which the cost of each arc between states in I is multiplied by w. As a consequence, for each state s in the interior of the update region I, the heuristic is updated using the following rule:

$$h(s) := \min_{t \in \partial I} w d_I(s,t) + h(t). \quad (3)$$

To produce wLSS-LRTA*, we simply change the implementation of Dijkstra's algorithm to consider a weighted cost function. To implement wLRTA*-LS, we do likewise but in addition we modify the way states are added to the learning space accordingly, by considering wc instead of c in the inequality used as a condition in the update step.

4. THEORETICAL ANALYSIS

In the previous section we proposed two techniques that perform RTHS using a heuristic h multiplied by a factor w greater than 1. Most RTHS algorithms have good properties (i.e., termination) when the heuristic h is consistent and remains consistent (see e.g., [10]). However, when heuristic h is consistent it is *not* necessarily the case that wh is consistent. Furthermore, we cannot guarantee that wh is admissible. Consequently, it is not obvious that good properties of a certain RTHS algorithm are inherited by its weighted version. In this section we analyze to what extent some important properties, like finding a solution when one exists, are preserved by our approaches even when the effective heuristic used during search may become inconsistent and hence inadmissible. In this paper, we focus on the properties of weighted update since this is the only technique that yields good empirical results.

As said above, we cannot ensure that the heuristic used in practice remains consistent, but can prove that it remains w-consistent. We furthermore, can guarantee that the heuristic will remain w-admissible if it is initially w-admissible. The definitions for w-consistency and w-admissibility follow.

Definition 1. *Given $w \geq 1$, we say h is w-consistent iff for each pair s, t of connected states $h(s) \leq h(t) + wc(s,t)$, and, for every goal state g, $h(g) = 0$.*

Definition 2. *Given $w \geq 1$, we say h is w-admissible iff for each $s \in S$ we have $h(s) \leq wh^*(s)$.*

Analogous to the case of regular consistency, given that h is w-consistent we can prove h is w-admissible. Henceforth, we assume wlog that there is a single goal g.

Theorem 1. *If h is w-consistent then h is w-admissible.*

Proof. Let $s \in S$, and let $\sigma = (s_1, s_2, ..., s_n)$, with $s_1 = s$ and $s_n = g$, be the shortest path from s to g. Since s_n is the goal state $h(s_n) = 0$. Furthermore, since $h(s_i) - h(s_{i+1}) \leq wc(s_i, s_{i+1})$, for every $i \in \{1, \ldots, n-1\}$ then

$$h(s) = h(s_0) - h(s_n) = \sum_{i=1}^{n-1} h(s_{i+1}) - h(s_i)$$
$$\leq w \sum_{i=1}^{n-1} c(s_i, s_{i+1}) = wc(\sigma) = wh^*(s).$$

■

We now turn our attention to prove that any algorithm that can (correctly) incorporate our weighted update will terminate. First we prove the following intermediate result.

Lemma 1. *Let h be a w-consistent heuristic. If we apply the Dijkstra algorithm in a learning space L then value of h will not decrease.*

Proof. Let us denote by h' the new heuristic function after running the weighted update Djikstra algorithm on region L. Note that $h = h'$ in $S \setminus L$. Let $\bar{L}$ be the set of all states in L whose h-value has decreased. We will prove that $\bar{L} = \emptyset$ by contradiction. Assume $\bar{L} \neq \emptyset$ and let $l \in \bar{L}$ be a state with minimum h' value in L. By correction of Dijkstra Algorithm any vertex $s \in L$ satisfies

$$h'(s) = \min_{t \in Succ(s)} h'(t) + wc(s,t),$$

Now let $u = \arg\min_{t \in Succ(l)} h'(t) + wc(l,t)$ then

$$h'(l) = h'(u) + wc(l,u) > h'(u).$$

We observe now that it should hold that $h(u) \leq h'(u)$, otherwise u would be in $\bar{L}$ and then we would use that $h'(l) > h'(u)$ to contradict that l is the state with minimal h'-value in $\bar{L}$. Using that $h'(u) \geq h(u)$ and that h is w-consistent we have

$$h'(l) = h'(u) + wc(l,u) \geq h(u) + wc(l,u) \geq h(l)$$

that contradicts the fact that $h'(l) < h(l)$. We conclude that $\bar{L}$ is empty and that no state in L decreases its h-value.

■

Now we establish that the property of w-consistency is preserved by the algorithm. The proof follows from the two following Lemmas.

Lemma 2. *If h is a w-consistent heuristic then h remains w-consistent after running a w-weighted update.*

Proof. Let $s \in S$. We have 3 cases:
Case 1: If $s \in I$ then

$$h'(s) = \min_{t \in Succ(s)} h'(t) + wc(s,t)$$

then for all t such that $(s,t) \in A$ we have that $h'(s) \leq h'(t) + wc(s,t)$.
Case 2: If $s \notin I$ and $s \neq g$ and $(s,t) \in A$. Since h is not updated outside of I, $h'(s) = h(s)$. Because h is consistent and $h(t) \leq h'(t)$ (by Lemma 2), the following inequality holds:

$$h'(s) = h(s) \leq h(t) + wc(s,t) \leq h'(t) + wc(s,t).$$

Case 3: if $s \notin I$ and $s = g$ then $h'(s) = h(s) = 0$ and we conclude that h' is a w-consistent heuristic function. ■

Lemma 3. *If the movement phase of an RTHS algorithm may only increase costs in the search graph, then w-consistency is preserved by the movement phase.*

Proof. Let c' denote the cost function after the movement phase. Since costs may only increase, $c \leq c'$. If $(s,t) \in A$ and h is w-consistent then $h(s) \leq wc(s,t) + h(t)$, which implies $h(s) \leq wc'(s,t) + h(t)$. ■

Theorem 2. *If h is initially w-consistent, then it remains w-consistency along the execution of an RTHS algorithm that uses a w-weighted update and whose movement phase may only increase the costs of arcs in the search graph, and whose lookahead phase does not change h or c.*

Table 1: Average solution cost and average total runtime for wLSS-LRTA*. Time is measured in milliseconds.

	$w = 1$		$w = 2$		$w = 4$		$w = 8$		$w = 16$		$w = 32$		$w = 64$	
k	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**
1	1,124,078	1,210.4	558,056	578.6	262,301	262.5	162,348	159.1	126,681	122.5	113,666	109.0	113,211	111.5
2	612,437	973.9	329,576	474.6	169,091	214.7	102,187	120.9	76,749	89.6	69,836	81.1	69,586	82.5
4	365,842	822.2	198,219	381.3	115,373	188.2	66,880	96.8	46,828	64.8	39,143	53.9	39,326	55.1
8	220,895	650.7	113,874	302.4	69,511	156.3	44,414	86.9	30,477	55.7	24,613	44.5	24,079	44.2
16	123,565	456.9	69,758	249.8	43,615	142.0	29,447	84.9	22,073	58.6	17,716	46.3	17,181	45.4
32	69,594	342.4	41,208	204.0	27,014	132.5	20,677	94.7	15,178	65.9	13,581	57.4	14,116	58.4
64	39,041	277.1	24,440	182.3	17,943	139.2	14,542	113.1	11,911	89.7	11,052	81.2	12,439	86.2
128	21,351	252.4	14,773	186.9	11,622	157.3	10,226	142.7	9,336	127.5	9,959	128.0	12,959	148.3

Table 2: Average solution cost and average total runtime for wLRTA*-LS. Time is measured in milliseconds.

	$w = 1$		$w = 2$		$w = 4$		$w = 8$		$w = 16$		$w = 32$		$w = 64$	
k	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**	**cost**	**time**
1	1,124,078	1,099.4	558,056	520.4	262,301	230.6	162,348	137.9	126,681	107.9	113,666	98.5	113,211	95.6
2	775,633	1,352.1	445,727	732.3	301,564	481.5	326,370	514.8	469,685	740.0	570,584	900.4	607,585	959.7
4	404,909	1,084.3	268,809	697.9	392,873	1,010.7	876,587	2,255.0	1,251,065	3,212.7	1,455,429	3,742.1	1,552,993	3,995.0
8	220,662	863.1	116,694	445.6	64,905	245.0	43,843	164.7	47,070	176.3	54,645	204.7	62,129	233.0
16	114,425	625.2	63,074	344.3	39,246	214.0	32,376	177.5	51,349	282.9	75,444	415.6	94,323	519.9
32	60,263	409.4	36,111	246.1	22,998	156.0	19,956	135.8	25,130	170.9	32,621	222.8	41,419	283.1
64	33,364	282.0	20,654	172.5	14,574	122.6	14,048	120.1	18,916	162.8	24,086	208.2	28,400	245.6
128	18,525	202.8	12,258	128.1	9,812	103.5	11,171	119.7	16,521	178.1	21,128	228.5	24,298	263.1

PROOF. Straightforward from Lemmas 2 and 3, and the fact that the lookahead phase does not change h or c. ■

Now we focus on our termination result, and assume we are dealing with an RTHS algorithm that satisfies the conditions of Theorem 2. For notational convenience, let h_n denote the heuristic function at the beginning of the n-th iteration of the RTHS algorithm. An important intermediate result is that eventually h converges.

LEMMA 4. *h eventually converges; that is, there exists an $N \in \mathbb{N}$ such that $h_{n+1} = h_n$ for all $n \geq N$.*

PROOF. Let c_* denote the minimum cost arc in (S, A). h_n is a bounded non decreasing series, thus by elementary calculus the series converges pointwise, that is, $h_n(s)$ converges for all $s \in S$. Moreover, if the $h(s)$ increased in some iteration n, then $h_{n+1}(s) - h_n(s) > wc_*$ and hence the h-value of s cannot increase more than $\frac{h^*(s)}{c_*}$ times. Convergence therefore is reached for a finite number N. ■

THEOREM 3. *Both wLSS-LRTA* and wLRTA*-LS reach g if the heuristic is initially w-consistent.*

PROOF. Suppose the assertion is false. Since by Lemma 4 h converges, at some iteration h does not change anymore and the agent enters a loop. Assume $\sigma = s_1, s_2, .., s_n, s_1$ is such a loop, and let $\sigma' = s'_1, s'_2, ..., s'_m, s'_1$ be the states at which the agent runs a lookahead step. Notice that $|\sigma'| \leq |\sigma|$ and that $\sigma' = \sigma$ for LRTA*-LS. Without loss of generality, assume s'_1 is one of the states of the loop with smallest heuristic value. Let L be the local search space of the algorithms. Since h does not change, there exists a state $s \in \partial L$ and such that $h(s'_1) = h(s) + wd_L(s'_1, s)$, otherwise h would be updated. (Note that this is still true for LRTA*-LS since $L = \{s'_1\}$ in this case.)

Since both wLRTA*-LS and wLSS-LRTA*, decide to move to the best state in ∂L, and that such a state is s'_2, we know that

$$h(s) + d_L(s'_1, s) \geq h(s'_2) + d_L(s'_1, s'_2), \qquad (4)$$

But we have that $h(s'_1) = h(s) + wd_L(s'_1, s)$. Substituting $h(s)$ in (4), we obtain:

$$h(s'_1) - wd_L(s'_1, s) + d_L(s'_1, s) \geq h(s'_2) + d_L(s'_1, s'_2). \qquad (5)$$

Finally, because s'_1 has a lowest h-value in σ, we have that $h(s'_1) \leq h(s'_2)$ and thus, using (5), we obtain:

$$h(s'_2) - wd_L(s'_1, s) + d_L(s'_1, s) \geq h(s'_2) + d_L(s'_1, s'_2).$$

Then,

$$-wd_L(s'_1, s) + d_L(s'_1, s) \geq d_L(s'_1, s'_2).$$

Rearranging, we obtain:

$$w \leq 1 - \frac{d_L(s'_1, s'_2)}{d_L(s'_1, s)} < 1,$$

which is a contradiction with the fact that $w \geq 1$. Thus, the agent cannot enter an infinite loop. ■

REMARK 1. *The last proof works on any algorithm whose movement decision is based in a greedy 'move to the best'.*

5. EVALUATION

We use eight-neighbor grids in the experiments since they are often preferred in practice, for example in video games [2]. The algorithms are evaluated in the context of goal-directed navigation in a priori unknown grids [16, 9]. The agent always senses the blockage status of its eight neighboring cells and can then move to any one of the unblocked neighboring cells with cost one for horizontal or vertical movements and cost $\sqrt{2}$ for diagonal movements. The user-given h-values are the octile distances [3].

We used twelve maps from deployed video games to carry out the experiments. The first six are taken from the game *Dragon Age*, and the remaining six are taken from the game *StarCraft*. The maps were retrieved from Nathan Sturtevant's pathfinding repository [15]. For Dragon Age we used the maps brc202d, orz702d, orz900d, ost000a, ost000t and ost100d of size 481×530, 939×718, 656×1491, 969×487,

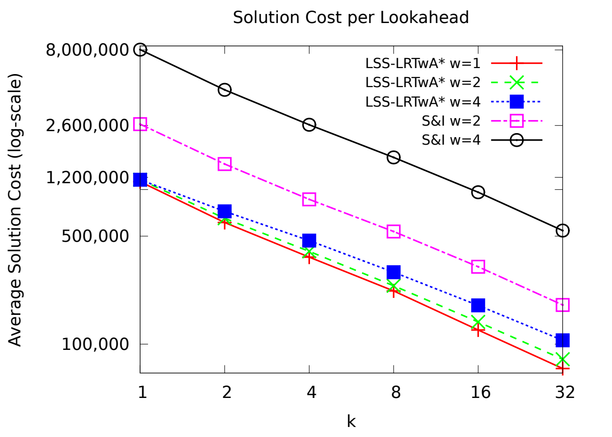

Figure 1: LSS-LRTwA* and Shimbo and Ishida's approach (S&I)

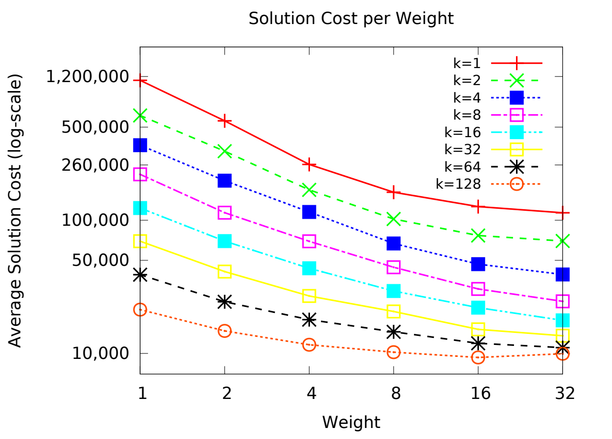

Figure 2: wLSS-LRTA*

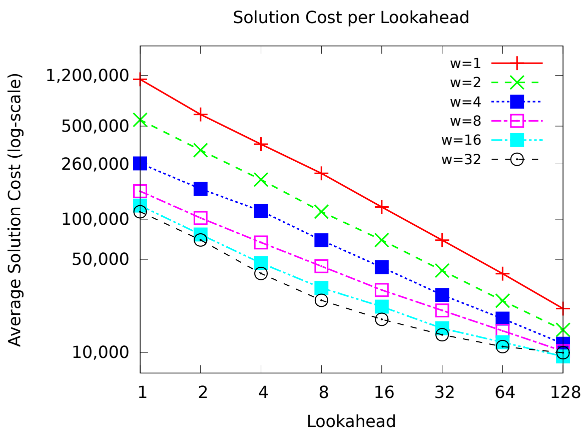

Figure 3: wLSS-LRTA*

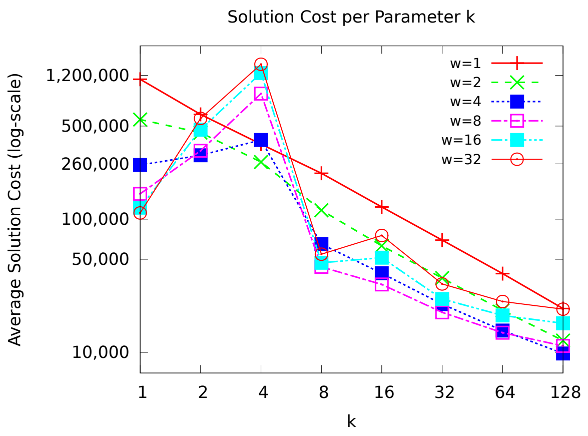

Figure 4: wLRTA*-LS

971×487, and 1025×1024 cells respectively. For StarCraft, we used the maps ArcticStation, Enigma, Inferno JungleSiege, Ramparts and WheelofWar of size 768×768, 768×768, 768×768, 768×768, 512×512 and 768×768 cells respectively. We average our experimental results over 200 test cases with a reachable goal cell for each map. For each test case the start and goal cells are chosen randomly. All the experiments were run in a 2.00GHz QuadCore Intel Xeon machine running Linux.

We evaluated LSS-LRTwA* and Shimbo and Ishida's approach for three weight values $\{1, 2, 4\}$ and six values for the lookahead parameter $\{1, 2, 4, 8, 16, 32\}$. The algorithms were implemented in a similarly, using a standard binary heap for the *Open* list and breaking ties among states with the same f-value in favor of larger g-values. Figure 1 shows a plot of solution cost versus lookahead parameter. We conclude that as w increases, the solution cost obtained by LSS-LRTwA* also increases. Larger differences are observed when the lookahead parameter increases. On the other hand, for Shimbo and Ishida's approach the solution cost increases more significantly when w is increased. We do not show a graph for times, but they look similar to Figure 1. We conducted a preliminary evaluation for LRTwA*-LS that showed a similar phenomenon.

We evaluated wLSS-LRTA* and wLRTA*-LS with six weight values $\{1, 2, 4, 8, 16, 32\}$ and eight lookahead values $\{1, 2, 4, 8, 16, 32, 64, 128\}$. As before, we used similar implementations. We report the solution cost obtained by the algorithms for different weight and lookahead values. Regarding the time per search episode, it is known that the time per search episode increases when the lookahead increases [5, 8]. On the other hand, when different weights are used for a fixed lookahead value, the time per search episode does not increase.

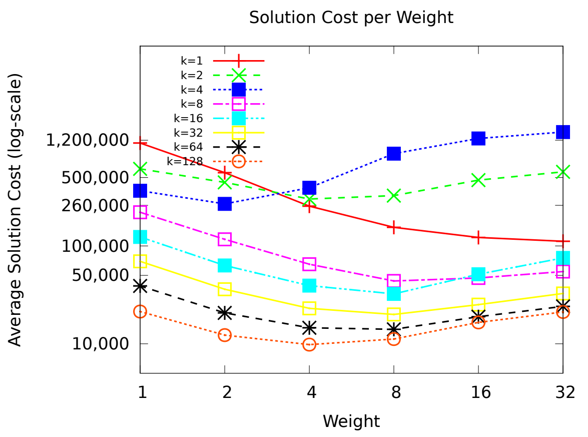

Figure 5: wLRTA*-LS

Figures 2 and 3, and Table 1 show the results for wLSS-LRTA*. The following can be observed in the plots:

- When the weight value increases the solution cost decreases for all lookahead values tested. Performance gains are more pronounced when the weight parameter is lower. For example, for the case of $k = 1$ we have that solution cost is reduced by 50% when increasing w from 1 to 2. On the other hand, the cost reduction is only .4% when increasing w from 32 to 64.
- When the lookahead value increases the solution cost decreases for all weight values tested.
- The plots do not show total search time because for all weight values tested the total search time behaves similar to the original algorithm (wLSS-LRTA* ($w = 1$))[7, 8]: first the total search time decreases when the lookahead value increases and from medium lookahead values the total search time smoothly increases when the lookahead value increases. On the other hand, when the weight value increases the total search time decreases.

Figures 5 and 4, and Table 2 show the results for wLRTA*-LS. The following can be observed in the plots:

- When the weight value increases the cost of the solution first decreases and from a certain weight value the solution cost increases slowly, for all lookahead values tested. For instance, for lookahead value equal to 16 the solution cost decreases until the weight equals 8 and then the solution cost increases.
- When the lookahead value increases the solution cost decreases for all weight values tested, except for some small lookahead values, for instance lookahead value equal to 4.
- For all weight values tested the total search time behaves similar to the original algorithm (wLRTA*-LS weight = 1)[5]: the total search time first decreases when the lookahead value increases, except for some small lookahead values where total search time increases, for instance lookahead value equal to 4. When the weight value increases the total search time decreases for small weight values, and from medium weight values the the total search time increases.

The results show that the use of weights in wLSS-LRTA* have very positive consequences. The solution cost and the total search time are improved sometimes by orders of magnitude without adding additional time in the search episodes.

We observed similar results for wLRTA*-LS but not for all weights and lookaheads tested. As the weight increases solution cost decreases but at certain point it starts increasing again. This could be due to the fact that h-values amplified by w are too high overestimate too much the actual heuristic values for a significant portion of the search space; in particular, for states that are closer to the goal. In the following section we discuss some insights of the performance of wLRTA*-LS.

6. DISCUSSION

Two of the results shown in the previous section could be seen as rather surprising. First and foremost is the fact that plugging Weighted A* – an algorithm that yields good results in Heuristic Search – into RTHS algorithms yields poor results. The second is the anomalous behavior exhibited by LRTA*-LS for some values of the k parameter. We analyze both of these issues below.

6.1 Weighted A* in RTHS

As seen in the last section, LSS-LRTwA* has very poor performance as w increases. Such finding is interesting, as it shows that the benefits of weighted A* cannot be immediately leveraged into RTHS.

Even though we do not have formal proofs that show why this happens we think two factors may play an important role. The first factor comes from a known property of Weighted A*: the solution cost typically increases as w increases. As such, it should not be surprising that worse intermediate solutions are returned by each of the calls in the lookahead step, which could explain why more costly solutions are found. Furthermore, since the number of expanded nodes in each search episode is *constant* (it is equal to the k parameter), using Weighted A* does not yield any time benefits either per lookahead step. Since the solution found is longer, more iterations of the RTHS algorithm are needed, which explains the increase in total time.

The poor performance of LSS-LRTwA* may also be explained by the quality of learning, which, at least in some parts of the learning space, is worse than when using LSS-LRTA*. In fact, assume that we want to construct a learning space of size k around state s. The next theorem states that the region built by expanding k nodes using A* is the one that *maximizes* the increase of the h-value of s. In other words, such a region maximizes learning in s.

THEOREM 4. *Let $k \in \mathbb{N}$, h be a consistent heuristic and $s \in S$ be such that $d(s,g) > k \max_{(s,t)\in A} c(s,t)$ (i.e., g is at least k steps away from s). Furthermore, let $\Delta_h(s,L) = \min_{t\in\partial L} h(t) + d_L(s,t) - h(s)$ and consider the following optimization problem*

$$\max_{L:s\in L\wedge|L|=k} \Delta_h(s,L).$$

Then the maximum is attained when L is the Closed list of an A search just after Closed reaches size k.*

Since Weighted A*, given a bound of k expansions, generates a region *different* from A*, we can infer that in some cases the learning performed in the current state is of inferior quality. This suggests that part of the poor performance of LSS-LRTwA* may be explained by its poor learning, since it is known that stronger learning yields better performance, all other things being equal (see e.g., [7]).

6.2 wLRTA*-LS for Low Values of k

Interestingly the decrease in the performance of wLRTA*-LS for $k = 2$ and $k = 4$ can be explained in very simple terms. Such a bad behavior should be expected whenever k is lower than the branching factor of the search problem. Indeed when that is the case, wLRTA*-LS will update the heuristic value of only *some* of the neighbors of the current state. Since in the movement phase wLRTA*-LS chooses the position to move to from its immediate *neighbors* it could be the case that the h-values of those neighbors are quite incomparable, because only some of them have been updated using w. In these situations it could be that the

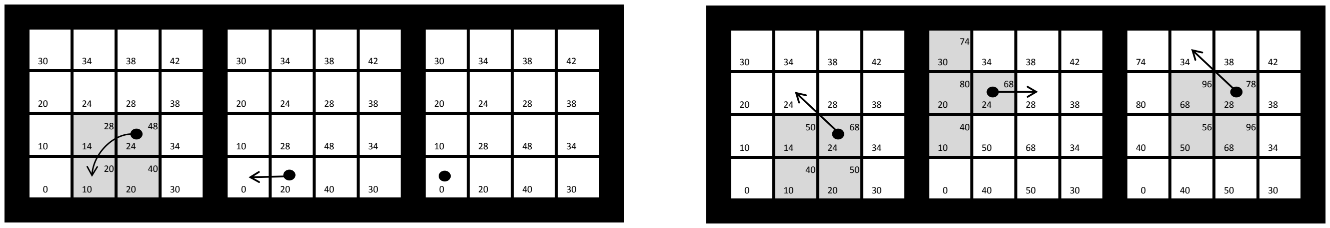

Figure 6: First three iterations of two runs of wLRTA*-LS with $w = 2$ (left) and $w = 4$ (right) with parameter $k = 4$ in a 4×4 grid. The grid is 8-connected, horizontal moves have cost 10, and diagonal movements have cost 14. The heuristic used is the obvious adaptation of the octile distance . Each cell shows the h-value before the update step in the lower-left corner, and the h-value after update in the upper right corner. The black dot shows the current position of the agent and the arrow shows the next cell chosen by the algorithm. We assume that states are added to the queue Q in clockwise order starting at 6 o'clock. The goal state is the state with heuristic value 0. We observe that when $w = 2$ it takes 2 movements to reach the goal. On the other hand, we observe that when $w = 4$ the agent moves away from the goal. In fact, when $w = 4$ it takes the agent 7 moves to reach goal.

algorithm chooses to move away from the goal. Figure 6 illustrates how this phenomenon may affect performance in 8-connected grid navigation.

wLSS-LRTA* does not have this problem, because it always chooses to move to the best state in *Open*. Since the h-values of those states is *not* updated, they are comparable. This observation suggests that wLRTA*-LS movement step could be modified in order to move to the state from ∂I that justifies the h-value of the current state. We decided to leave the implementation of such an algorithm out of the scope of this paper. We conjecture that it will not exhibit performance degradation for low values of k.

7. SUMMARY

We proposed two approaches that allow exploiting weights in RTHS algorithms. We showed that the approaches that incorporates Weighted A* within RTHS algorithms yields poor performance, whereas the technique that incorporates weights in the update phase yields superior performance of up to one order of magnitude.

Our technique is applicable to a wide range of RTHS algorithms that use A* in the lookahead and Dijkstra's algorithm for the update. How to include our technique in algorithms such as RTAA*, which uses a quite different update rule, is not obvious and remains as future work.

Acknowledgments.
Jorge Baier was partly funded by Fondecyt grant number 11110321.

8. REFERENCES

[1] V. Bulitko. Learning for adaptive real-time search. *Computing Research Repository*, cs.AI/0407016, 2004.

[2] V. Bulitko, Y. Björnsson, N. Sturtevant, and R. Lawrence. *Real-time Heuristic Search for Game Pathfinding.* Applied Research in Artificial Intelligence for Computer Games. Springer, 2011.

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

[4] 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):100–107, 1968.

[5] C. Hernández and P. Meseguer. Improving LRTA*(k). In *Proc. of the 20th Int'l Joint Conf. on Artificial Intelligence (IJCAI)*, pages 2312–2317, 2007.

[6] C. Hernández and J. A. Baier. Avoiding and escaping depressions in real-time heuristic search. *Journal of Artificial Intelligence Research*, 43:523–570, 2012.

[7] S. Koenig and M. Likhachev. Real-time adaptive A*. In *Proc. of the 5th Int'l Joint Conf. on Autonomous Agents and Multi Agent Systems (AAMAS)*, pages 281–288, 2006.

[8] 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.

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

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

[11] M. Likhachev, G. J. Gordon, and S. Thrun. ARA*: Anytime A* with Provable Bounds on Sub-Optimality. In *Proc. of the 16th Conf. on Advances in Neural Information Processing Systems (NIPS)*, Vancouver, Canada, 2003.

[12] I. Pohl. Heuristic search viewed as path finding in a graph. *Artificial Intelligence*, 1(3):193–204, 1970.

[13] S. Richter, J. T. Thayer, and W. Ruml. The joy of forgetting: Faster anytime search via restarting. In *Proc. of the 20th Int'l Conf. on Automated Planning and Scheduling (ICAPS)*, pages 137–144, 2010.

[14] M. Shimbo and T. Ishida. Controlling the learning process of real-time heuristic search. *Artificial Intelligence*, 146(1):1–41, 2003.

[15] N. R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[16] A. Zelinsky. A mobile robot exploration algorithm. *IEEE Transactions on Robotics and Automation*, 8(6):707–717, 1992.