Proceedings of the Ninth International
Symposium on Combinatorial Search (SoCS 2016)

Weighted Lateral Learning in Real-Time Heuristic Search

Vadim Bulitko and **Alexander Sampley**
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
{bulitko|sampley}@ualberta.ca

Abstract

Real-time heuristic search models an autonomous agent solving a search task. The agent operates in a real-time setting by interleaving local planning, learning and move execution. In this paper we propose a simple parametric algorithm that combines weighting with learning from multiple neighbors. Doing so breaks heuristic admissibility but allows the agent to escape heuristic depressions more quickly. We prove completeness of the algorithm and empirically compare it to several competitors more than twenty years apart. In a large-scale evaluation the new algorithm found better solutions than the recent algorithms, despite not learning additional information that they do. Finally, we study robustness of the algorithms to noise in the heuristic function — a desirable property in a physical implementation of real-time heuristic search. The new algorithm outperforms its contemporaries.

1 Introduction and Related Work

Real-time heuristic search is a sub-area of heuristic search which deals with agent-centered search under time constraints. It models an agent with locally limited sensing and perception that is trying to reach a goal (Koenig 2001). The *real-time constraint* forces the agent to produce each action in a constant-bounded amount of time regardless of the total number of states in the search problem. The constraint excludes classical algorithms such as A* (Hart, Nilsson, and Raphael 1968) and its more recent variants such as PRA* (Sturtevant and Buro 2005; Sturtevant 2007) but allows for its real-time version TBA* (Björnsson, Bulitko, and Sturtevant 2009).

Starting with the seminal work by Korf (1990) such agents usually interleave three processes: local planning, heuristic learning and move selection. Researchers have explored different methods for looking ahead during the planning stage of each cycle (Koenig and Sun 2009); different heuristic learning rules (Bulitko 2004; Hernández and Meseguer 2005; Bulitko and Lee 2006; Rayner et al. 2007; Koenig and Sun 2009; Rivera, Baier, and Hernández 2015) and different move selection mechanisms (Ishida 1992; Shue and Zamani 1993a; 1993b; Shue, Li, and Zamani 2001; Bulitko and Lee 2006; Hernández and Baier 2012). Finally, information in addition to the heuristic has been learned during (Bulitko et al. 2007; Sturtevant, Bulitko, and Björnsson 2010; Sturtevant and Bulitko 2011; Sharon, Sturtevant, and Felner 2013) and before (Bulitko et al. 2008; Bulitko, Björnsson, and Lawrence 2010; Botea 2011; Bulitko, Rayner, and Lawrence 2012; Lawrence and Bulitko 2013) the search.

In this paper we focus on the learning rule for the heuristic function and make the following contributions. First, we describe a simple parameterized algorithm that spans the space between LRTA* (Korf 1990) and depression-avoiding algorithms such as aLRTA* (Hernández and Baier 2012). It does so in a principled way by using two continuous control parameters whose influence we study in this paper. Second, we prove that for any valid parameter values, our algorithm will indeed solve the problem despite learning an inadmissible heuristic. Third, we conduct a large scale empirical evaluation using the standard pathfinding benchmarks (Sturtevant 2012). We evaluate our algorithm against recent competitors with two practically important performance measures: first-trial solution cost and the amount of state revisitation. The results show that the new algorithm produces better solutions than its modern competitors despite not using additional information such as marking depressed states in aLRTA* (Hernández and Baier 2012) or expendable states in daLRTA*+E (Sharon, Sturtevant, and Felner 2013) or using the combination of heuristic weighting and heuristic update magnitude in wdaLRTA* (Rivera, Baier, and Hernández 2015). Fourth, we measure how robust the high-performance algorithms are to input/output errors in the heuristic function and observe that our algorithm appears to be more robust than its competitors.

While the combination of lateral learning and weighted updates was briefly presented and evaluated in our recent conference publication (Bulitko 2016a) and its follow-up (Bulitko 2016b), this paper motivates both mechanisms with examples, proves their theoretical properties and evaluates them in the presence of heuristic access errors. We do re-use the problem definition and some of the experimental set-up from (Bulitko 2016a). Accordingly, we reproduce the relevant parts from that paper.

2 Problem Formulation

In line with previous research, we define a *search problem* **S** as the tuple (S, E, c, s_0, s_g, h) where S is a finite set of *states* and $E \subset S \times S$ is a set of *edges* between them. S and E jointly define the search graph which is assumed undirected: $\forall a, b \in S\,[(a,b) \in E \implies (b,a) \in E]$ and has no self-loops: $\forall s \in S\,[(s,s) \notin E]$. The graph is weighted by the strictly positive edge costs $c : E \to \mathbb{R}^+$. Two states a and b are *immediate neighbors* iff there is an edge between them: $(a,b) \in E$; we denote the set of immediate neighbors of a state s by $N(s)$. A *path* P is a sequence of states $(s_0, s_1, \dots, s_n)$ such that for all $i \in \{0, \dots, n-1\}$, $(s_i, s_{i+1}) \in E$. We assume that the search graph (S, E) is connected (i.e., any two vertices have a path between them) which makes it safely explorable. We also assume that the search graph is stationary (e.g., the edge weights do not change during search).

At all times $t \in \{0, 1, \dots\}$ the agent occupies a single state $s_t \in S$, called the *current state*. The state s_0 is the start state and is given as a part of the problem. The agent can change its current state, that is, move to any immediately neighboring state in $N(s)$. The traversal incurs a travel *cost* of $c(s_t, s_{t+1})$. The agent is said to solve the search problem at the earliest time T it arrives at the goal state: $s_T = s_g$. The *solution* is a path $P = (s_0, \dots, s_T)$: a sequence of states visited by the agent from the start state until the goal state. The cumulative cost of all edges in the solution is called *solution cost* and is formally defined as: $c_A(\mathbf{S}) = \sum_{t=0}^{T-1} c(s_t, s_{t+1})$ for algorithm A. The cost of the shortest possible path between states $a, b \in S$ is denoted by $h^*(a, b)$. We abbreviate $h^*(s, s_g)$ as $h^*(s)$. We define *suboptimality* of the agent on a problem as the ratio of the solution cost the agent incurred to the cost of the shortest possible solution: $\alpha(A, \mathbf{S}) = {c_A(\mathbf{S})}/{h^*(s_0)}$. For instance, suboptimality $\alpha(\text{LRTA*}, \mathbf{S}) = 2$ means that the agent driven by the LRTA* algorithm found a solution to $\mathbf{S}$ twice as long as optimal. Lower values are preferred; 1 indicates optimality.

The other performance measure we are concerned with in this paper is the *scrubbing complexity* (Huntley and Bulitko 2013). It is defined as the average number of state visits the agent makes while solving a problem. Formally, let $v_A^{\mathbf{S}} : S \to \mathbb{N} \cup \{0\}$ be the number of state visits the agent driven by algorithm A made while solving a problem $\mathbf{S}$. The scrubbing complexity is then defined over the subset of states that the agent visited at least once: $S_{\text{visited}} = \{s' \in S \mid v_A^{\mathbf{S}}(s') \geq 1\}$: $\tau(A, \mathbf{S}) = \frac{1}{|S_{\text{visited}}|} \sum_{s \in S_{\text{visited}}} v_A^{\mathbf{S}}(s)$. For instance, $\tau(\text{LRTA*}, \mathbf{S}) = 7.5$ means that while solving problem $\mathbf{S}$, on average the agent driven by the LRTA* algorithm visited a state 7.5 times (states that were not visited at all do not contribute to the average). Lower values of $\tau(\mathbf{S})$ are preferred since re-visiting states tends to look irrational to an external observer. This is a major reason why real-time heuristic-search methods are hardly used for pathfinding in actual video games. Instead, game developers prefer non-real-time heuristic search algorithms such as variants of path-refinement A* (Sturtevant 2007).

In its operation the agent has access to a heuristic $h : S \to [0, \infty)$. The heuristic function is a part of the search problem specification and is meant to give the agent an estimate of the remaining cost to go. Unlike much literature in the field, we do *not* assume admissibility or consistency of the initial heuristic but require that $h(s_g) = 0$. The search agent can modify the heuristic as it sees fit as long as it remains non-negative and the heuristic of the goal state s_g remains 0. The heuristic at time t is denoted by h_t; $h_0 = h$.

We say that a search agent is *real time* iff its computation time between its moves is upper-bounded by a constant independent of the number of states in the search space. We will additionally require our search algorithms to be *agent centered* insomuch as they have access to the heuristic, states and edges only in a bounded vicinity of the agent's current state and the bound is independent of the number of states (Koenig 2001). We say that a search agent is *complete* iff it solves any search problem as defined above. That is, it is required to terminate in the goal state s_g at time $T < \infty$.

The problem we tackle in this paper is to develop a real-time heuristic search algorithm that is complete and has low suboptimality (α) and low scrubbing complexity (τ).

3 Framework

As Section 1 presented, many ways of improving on LRTA* towards the two measures have been proposed. In this paper we specifically focus on heuristic learning rules. To isolate the problem, we fix the lookahead at 1 (i.e., allow the agent to consider only the immediate neighbors of its current state during the planning stage), set the move rule to that of LRTA* (Korf 1990) and allow the agent to update its heuristic only in its current state. In other words, our local search space and the local learning spaces are limited to the agent's current state. With these limitations in place, LRTA* can be described as Algorithm 1.

Algorithm 1: Basic Real-time Heuristic Search

```
input  : search problem $(S, E, c, s_0, s_g, h)$
output: path $(s_0, s_1, \dots, s_T), s_T = s_g$
1 $t \leftarrow 0$
2 $h_t \leftarrow h$
3 while $s_t \neq s_g$ do
4     $s_{t+1} \leftarrow \arg\min_{s \in N(s_t)} (c(s_t, s) + h_t(s))$
5     $h_{t+1}(s_t) \leftarrow \max\left\{h_t(s_t),\ \min_{s \in N(s_t)} (c(s_t, s) + h_t(s))\right\}$
6     $t \leftarrow t + 1$
7 $T \leftarrow t$
```

A search agent following the algorithm begins in the start state s_0. It then executes a fixed loop until it reaches the goal s_g (line 3). At each iteration of the loop, the agent expands the current state s_t by generating its immediate neighbors $N(s_t)$ (local planning). It computes its action by selecting the next state s_{t+1} among the neighbors to minimize the estimated cost of traveling to the goal through that neighbor (line 4). Ties among neighbors that have the same $c + h$ values are broken with a tie-breaking schema that is consistent over state revisits. Then, in line 5, the agent updates (learns) its heuristic in the current state from $h_t(s_t)$ to $h_{t+1}(s_t)$. Note that the explicit maximum of the state's old heuristic value and the new value causes the heuristic to never decrease. Such a maximum is unnecessary if the heuristic is consistent and is commonly omitted in the literature. We do not assume our heuristic to be consistent and hence put the maximum in explicitly. The agent then changes its state to the neighbor (i.e., makes a move) and the cycle repeats.

4 Our Approach

4.1 Heuristic Depressions

A major problem with real-time heuristic search algorithms is getting caught in heuristic depressions – areas of the state space where the heuristic values are inaccurately low. This problem was identified early on (Ishida 1992) and linked to state-revisitation and increased solution cost (Huntley and Bulitko 2013). Since early days of the field the problem has been addressed in a variety of ways. Some algorithms attempted to escape heuristic depressions via a deeper lookahead (Korf 1990; Koenig and Sun 2009). Other researchers suggested backtracking upon learning (Shue and Zamani 1993a; 1993b; Shue, Li, and Zamani 2001; Bulitko and Lee 2006) or learning in states beyond the current state (Hernández and Meseguer 2005; Rayner et al. 2007). Yet other methods learned additional information during the search (e.g., building a smaller abstracted search space (Bulitko et al. 2007) or marking certain states (Sturtevant, Bulitko, and Björnsson 2010; Sturtevant and Bulitko 2011; Hernández and Baier 2012; Sharon, Sturtevant, and Felner 2013)). Our focus on the learning rule in this paper precludes most of such depression-avoidance mechanisms and nearly brings us to provably unavoidable state revisitation (Sturtevant and Bulitko 2014).

4.2 Scaling Edge Costs → Faster Learning

Fortunately, unlike Sturtevant and Bulitko (2014), our framework does allow an agent to learn more aggressively than consistency/admissibility of the heuristic would allow. As argued by Rivera, Baier, and Hernández (2015), rapidly increasing the heuristic in the visited states discourages the agent from state-revisitation even without explicitly tracking previously visited states (Hernández and Baier 2012) or attempting to eliminate expendable states (Sharon, Sturtevant, and Felner 2013). In our framework, the weighting mechanism of Rivera, Baier, and Hernández (2015) reduces to replacing the update rule

$$h_{t+1}(s_t) \leftarrow \max\left\{h_t(s_t),\ \min_{s\in N(s_t)}(c(s_t,s)+h_t(s))\right\} \tag{1}$$

in line 5 of Algorithm 1 with

$$h_{t+1}(s_t) \leftarrow \max\left\{h_t(s_t),\ \min_{s\in N(s_t)}(w\cdot c(s_t,s)+h_t(s))\right\} \tag{2}$$

where the weight $w \geq 1$ is the control parameter. For $w > 1$, the heuristic update can break consistency and admissibility of the heuristic.

Note that this is a different approach than scaling up the initial heuristic (Shimbo and Ishida 2003) or scaling down the costs (Bulitko 2004; Bulitko and Lee 2006). Indeed, multiplying the initial heuristic h_0 by an appropriately chosen weight $w > 1$ will bring it closer to the optimal heuristic h^* in some states, thereby possibly reducing the volume of the heuristic depressions. Likewise, scaling down the edge costs effectively lowers h^*, therefore bringing the initial heuristic h_0 closer to it and producing a similar effect. These techniques can reduce convergence travel at the cost of increasing the first-trial solution cost. In contrast, scaling up the edge costs in the learning rule raises the heuristic in the visited states (possibly substantially above h^*) thereby discouraging the agent from revisiting those states. This has been shown to reduce the (first-trial) solution cost in practice (Rivera, Baier, and Hernández 2015).

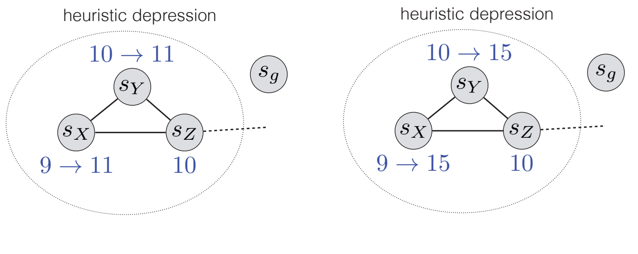

Figure 1: **Left:** LRTA* first updates $h(s_X)$ from 9 to $1 + 10 = 11$ and moves from s_X to s_Y. It then updates $h(s_Y)$ from 10 to $1 + 10 = 11$. **Right**: wLRTA* with $w = 5$ first updates $h(s_X)$ from 9 to $5 \cdot 1 + 10 = 15$ and moves from s_X to s_Y. It then updates $h(s_Y)$ from 10 to $5 \cdot 1 + 10 = 15$. All edge costs are 1.

To illustrate, consider the agent in a heuristic depression that is about to update the heuristic of its current state s_X (Figure 1, left). If it uses the regular consistency-preserving learning rule then $h(s_X)$ can only be raised to lowest $f = c + h$ of the immediate neighbors (s_Z in the figure). This is a modest raise and since the heuristic is raised only in the current state, many state revisits are necessary to "fill in" the heuristic depression (assume that the state on the other end of the dashed edge has a high h). Scaling the update by $w > 1$ (i.e., raising the heuristic of s_X to the minimum $w \cdot c + h$ among the neighbors) will speed up filling the heuristic depression and reduce the number of state re-visits (Figure 1, right).

4.3 Lateral Learning → Even Faster Learning

Note that when the agent arrives at s_Y, the raise to $h(s_Y)$ will *not* be affected by the previously raised $h(s_X)$ due to s_Z being a neighbor of both s_X and s_Y. This is because the update rule used by Rivera, Baier, and Hernández (2015) still uses the min operator and thus will link $h(s_Y)$ to the lowest of the neighbors (i.e., s_Z), ignoring already raised $h(s_X)$. The min operator is necessary to make sure $h_t \rightarrow w \cdot h^*$ over time. However, as shown in this example, it can slow down the propagation of heuristic raises and thus slow down escaping a heuristic depression.

In this paper we address this issue by replacing the min operator with the avg operator which allows the agent to learn not only from the lowest-f neighbor but from other neighbors as well. To do so we simply replace the min with the avg in the learning rule:

$$h_{t+1}(s_t) \leftarrow \max\left\{h_t(s_t),\ \operatorname*{avg}_{s\in N(s_t)}(w\cdot c(s_t,s)+h_t(s))\right\}. \tag{3}$$

In our running example, upon getting to state s_Y, the agent will raise its heuristic to the average of $h(s_Z)$ (still low) and $h(s_X)$ (already high) which will lead to a higher new h than merely the weighted update. In other words, replacing the $\min$ operator with avg allows the agent to learn *laterally* from non-minimum-f neighbors (Figure 2, left).[*]

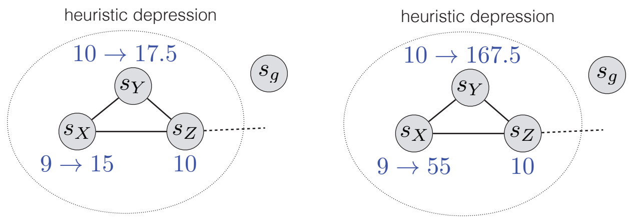

Figure 2: **Left:** wLRTA* with $w = 5$ and lateral learning first updates $h(s_X)$ from 9 to $\mathrm{avg}\{5\cdot1+10, 5\cdot1+10\} = 15$ and moves from s_X to s_Y. It then updates $h(s_Y)$ from 10 to $\mathrm{avg}\{5\cdot1+10, 5\cdot1+15\} = 17.5$. **Right**: our new algorithm with $w = 5$ first updates $h(s_X)$ from 9 to $\mathrm{avg}\{5\cdot(1+10), 5\cdot(1+10)\} = 55$ and moves from s_X to s_Y. It then updates $h(s_Y)$ from 10 to $\mathrm{avg}\{5\cdot(1+10), 5\cdot(1+55)\} = 167.5$.

The learning rule (3) combines edge-cost weighting with the lateral learning. We experimented with Algorithm 1 whose learning rule was replaced with the weighted lateral rule (4) and found it to perform very similarly to the recently proposed high-performance weighted depression-avoiding wdaLRTA* of Rivera, Baier, and Hernández (2015). This is interesting since wdaLRTA* has an explicit control mechanism in its move rule to attempt to get out of a heuristic depression. To do so it must keep the magnitude of heuristic updates (i.e., $h_t - h_0$) for all states. Our lateral learning appears to play an equivalent role in getting an agent to leave heuristic depressions quickly.

4.4 Weighted Lateral LRTA*

Note that the weighted lateral learning rule (3) applies the weight only to the edge cost. Even more aggressive learning can be obtained if the weight applies to both the edge cost and the heuristic. In other words, instead of raising h of the current state to $w \cdot c + h$, we can raise it to $w \cdot (c + h)$. This is illustrated in Figure 2, right.

We implement this idea in our new Algorithm 2. To make the new learning rule a generalization of existing rules, we can take the average over a *part* of the neighborhood like so:

$$h_{t+1}(s_t) \leftarrow \max \left\{ h_t(s_t),\ w \cdot \operatorname*{avg}_{s \in N_b^f(s_t)} (c(s_t, s) + h_t(s)) \right\}. \tag{4}$$

Here the partial neighborhood N_b^f of a state s_t is defined as the b fraction of the neighborhood $N(s_t)$ with the lowest f values:

$$N_b^f(s) = \left(s^1, \dots, s^{\lfloor b|N(s_t)|\rfloor}\right) \tag{5}$$

where $\left\{s^1, \dots, s^{\lfloor b|N(s_t)|\rfloor}, \dots, s^{|N(s_t)|}\right\}$ is the immediate neighborhood sorted[†] in the ascending order by their $f = c + h$. For instance, s^1 has the lowest $f(s^1) = c(s_t, s^1) + h(s^1)$ value in the set $\{f(s) \mid s \in N(s_t)\}$ whereas $s^{|N(s_t)|}$ has the highest f value in that set.

We also constrain all raises of the heuristic to be of at least μ in magnitude (line 7) where $\mu \geq 0$ is a control parameter (see Section 5 for the rationale).

Algorithm 2: Weighted Lateral LRTA* (wbLRTA*)

```
input : search problem $(S, E, c, s_0, s_g, h)$, control
        parameters: weight $w$, beam width $b$, minimum update
        $\mu$
output: path $(s_0, s_1, \dots, s_T)$ such that $s_T = s_g$
1 $t \leftarrow 0$
2 $h_t \leftarrow h$
3 while $s_t \neq s_g$ do
4     $s_{t+1} \leftarrow \arg\min_{s \in N(s_t)} (c(s_t, s) + h_t(s))$
5     $h_{t+1}(s_t) \leftarrow$
        $\max \left\{ h_t(s_t),\ w \cdot \operatorname{avg}_{s \in N_b^f(s_t)} (c(s_t, s) + h_t(s)) \right\}$
6     if $h_{t+1}(s_t) > h_t(s_t)$ then
7         $h_{t+1}(s_t) \leftarrow \max \{h_{t+1}(s_t), h_t(s_t) + \mu\}$
8     $t \leftarrow t + 1$
9 $T \leftarrow t$
```

Clearly, for $b = 1$ we get the full neighborhood: $N_1^f(s_t) = N(s_t)$. For $b = 0$ we define $N_0^f(s_t) = \{s^1\}$. We call b the *beam width*. Note that unlike the classic beam search and its extensions (Furcy and Koenig 2005), our beam width is used only in the learning rule, not to control the open list. Consequently, for $b = 0, w = 1$, line 5 in Algorithm 2 becomes:

$$h_{t+1}(s_t) \leftarrow \max \left\{ h_t(s_t),\ \operatorname*{avg}_{s = s^1} (c(s_t, s) + h_t(s)) \right\} \tag{6}$$

which is equivalent to line 5 in the original Algorithm 1:

$$h_{t+1}(s_t) \leftarrow \max \left\{ h_t(s_t),\ \min_{s \in N(s_t)} (c(s_t, s) + h_t(s)) \right\}. \tag{7}$$

Thus we just proved:

Lemma 1 wbLRTA* with $b = 0, w = 1, \mu = 0$ is equivalent to LRTA* (with a lookahead of 1).

Everywhere below we will assume the control parameters to be in their valid ranges: beam width $b \in [0, 1]$, weight $w \in [1, \infty)$ and the minimum update $\mu \in (0, \infty)$.

5 Theoretical Analysis

We will now show that wbLRTA* solves any search problem defined in Section 2. The original proof (Korf 1990) implied that infinitely many updates to the heuristic value of a state will lead to unbound heuristic growth. While this is true

[*]This may appear similar to bidirectional pathmax (BPMX) of Felner et al. (2011) except we break admissibility by *adding* costs and we *average* over neighbors' weighted f costs.

[†]We abuse the set notation as elements in a set have no order.

when the update rule is based on the minimum (line 5 in Algorithm 1), our update rule is based on averaging over possibly partial neighborhoods (line 5 in Algorithm 2). Thus, it is not immediately clear that the magnitude of the updates is lower-bounded by a positive constant (i.e., the averaging could cause magnitude of the updates tend to zero, breaking the unbounded growth of the heuristic). We constrain all raises of the heuristic to be of at least $\mu > 0$.

Another proof of completeness which covers a more general algorithm LRTS (Bulitko and Lee 2006) relied on admissibility of the heuristic at all times. This was also the case for depression avoidance algorithms (Hernández and Baier 2012). Weighted admissibility was used by Rivera, Baier, and Hernández (2015). Our algorithm hinges on aggressive heuristic learning that breaks (weighted) admissibility and hence their proofs may not apply. Thus we offer an alternative proof which also applies to the standard LRTA* (with a lookahead of 1). Yet another proof of completeness covered the case of inadmissible and inconsistent heuristics and even moving goal but assumed the standard LRTA*-style learning rule with the $\min$ operator (Shimbo and Ishida 2000).

We start with the following lemma (all proofs are found in Section 9 and all control parameters are assumed to be in the valid ranges specified above):

Lemma 2 Whenever an agent travels from state s_t to state s_{t+1}, the arrival heuristic of the latter state is at least ϵ less than the departing heuristic of the former state. Here $\epsilon > 0$ does not depend on the state.

Furthermore, every raise to a heuristic value is lower bounded by a finite constant:

Lemma 3 Whenever an agent moves from a state s_t to a state s_{t+1} the heuristic value of state s_t is either unchanged or raised by at least μ.

We can now prove that revisiting states infinitely many times leads to unbounded heuristic growth:

Lemma 4 For any $b \in [0, 1]$, $w \in [1, \infty)$, wbLRTA* will unboundedly raise heuristic values of all states in any $S' \subset S$ if (i) the agent is only visiting the states in S' after some initial time t': $\forall t > t' \, [s_t \in S']$ and (ii) each of the states in S' is visited infinitely many times: $\forall s' \in S' \, [|\{t \mid s_t = s'\}| = \infty]$.

We can now prove completeness of wbLRTA*.

Theorem 1 wbLRTA* is complete for any beam width $b \in [0, 1]$ and any weight $w \in [1, \infty)$.

6 Empirical Evaluation

Following the tradition in the field, we evaluated wbLRTA* in the *de facto* standard benchmark of real-time heuristic search algorithms: grid-based pathfinding. The standard way (Sturtevant 2012) of representing game maps is as a two-dimensional discrete grid where each grid cell is either available for the agent to pass through or blocked by an obstacle. At each moment of time, the agent occupies a single vacant cell of the map which determines the agent's current state. The agent changes its state by moving from its current grid cell to one of its vacant neighbors, incurring a travel cost. In this paper we use the standard eight-connected maps where cardinal moves cost 1 and diagonal moves cost $\sqrt{2}$. A problem is solved when the agent enters the goal cell. At the beginning of each problem, the agent starts with the octile distance as the heuristic. Octile distance is the cost of the shortest path between a given cell and the goal cell *if* no cells are blocked by obstacles.

We downloaded the benchmark problems from the Moving AI set (Sturtevant 2012) and treated the water terrain type as an obstacle. All problems that became unsolvable (e.g., the start state is in a water cell) were excluded. This gave us 493298 problems situated on 342 maps from the video games *StarCraft*, *WarCraft III*, *Baldur's Gate II* (maps scaled up to 512×512) and *Dragon Age: Origins*. We evaluated algorithms on sample sets of problems randomly drawn from the overall set of 493298 MovingAI problems.

6.1 Algorithm and Parameter Selection

We selected the following three algorithms as competitors to wbLRTA*: LRTA*, RTA* (Korf 1990) and the weighted variant aLRTA* (Rivera, Baier, and Hernández 2015). All of those conform to our framework by conducting a lookahead of 1, learning only in the current state and using the original rule for move selection. We tuned the algorithm parameters by running preliminary experiments (Bulitko 2016a) which we re-describe below for the reader's convenience. To select the weight in wLRTA*, we ran $w \in \{1, 2, 3, 4, 5, 6, 7, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$ on 6000 problems each and found that $w = 128$ gave the lowest suboptimality.

To select control parameters for our algorithm wbLRTA* we tried the beam width $b \in \{0, 0.1, 0.2, 0, 3, 0.4, 0.5, 0.6, 0.7, 0.8\}$ and the weight $w \in \{1, 2, \dots, 8\}$. Each combination was run on 6000 problems. The lowest mean suboptimality of 24.68 was achieved with $b = 0.6, w = 4$. The minimum update μ appeared to have no significant effect on suboptimality and was fixed at $\mu = 0.001$.

We also included several existing algorithms that, while breaking an assumption of our framework, all aim to escape heuristic depressions quicker than LRTA*. Specifically, we evaluated the myopic versions (lookahead of 1) of aLRTA*, daLRTA* (Hernández and Baier 2012), its weighted version wdaLRTA* (Rivera, Baier, and Hernández 2015) and its combination with expendable states, daLRTA*+E (Sharon, Sturtevant, and Felner 2013). We did not include f-LRTA* or f-LRTA*+E in our evaluation since both versions were found inferior to daLRTA* by Sharon, Sturtevant, and Felner (2013). To select the weight parameter for wdaLRTA*, we run $w \in \{1, 2, 3, 4, 5, 6, 7, 8, 16, 32, 64, 128, 256, 512, 1024, 2048\}$ on 6000 problems each and found that $w = 7$ gave the lowest suboptimality.

6.2 Results

We ran eight algorithms, each on 30000 problems randomly selected from the benchmarks. For each algorithm we measured its suboptimality α and scrubbing τ. The means are found in Table 1. Figure 3 zooms in on better algorithms.

Table 1: Performance of the algorithms (sample mean ± standard error of the mean).

Algorithm	**Suboptimality** α	**Scrubbing** τ
LRTA*	451.2 ± 7.157	16.28 ± 0.143
RTA*	353.9 ± 5.886	11.81 ± 0.101
aLRTA*	352.0 ± 6.329	11.19 ± 0.117
daLRTA*	45.6 ± 0.744	2.00 ± 0.011
wLRTA*	39.6 ± 0.534	1.84 ± 0.008
wdaLRTA*	30.7 ± 0.444	1.47 ± 0.005
daLRTA*+E	32.5 ± 0.906	1.21 ± 0.004
wbLRTA*	26.7 ± 0.431	1.28 ± 0.003

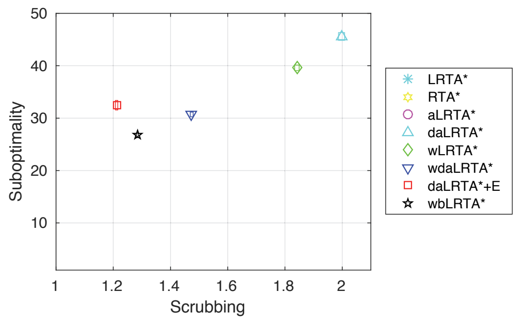

Figure 3: Sample means and standard errors of the mean shown as error boxes. LRTA*, RTA*, aLRTA* aren't shown.

Our new algorithm wbLRTA* appears to produce shorter solutions than the other algorithms, outperforming even aLRTA*, daLRTA*, wdaLRTA* and daLRTA*+E that go beyond our framework and learn information in addition to the heuristic function. In terms of scrubbing, wbLRTA* takes the second place, slightly behind daLRTA*+E.[‡] These are promising results for the new weighted lateral learning rule.

6.3 Robustness to Heuristic Access Errors

The new algorithm appears to produce shorter solutions than its modern competitors in the standard pathfinding testbed of real-time heuristic search algorithm. The fundamental assumption of all such algorithms is that the heuristic can be stored precisely and reliably. In order to do so, in a typical implementation the agent holds the entire heuristic in its memory. So while real-time heuristic search assumes that the agent's response time does not scale up with the number of states in the search graph, it tacitly requires the agent's memory capacity to grow indefinitely with the environment size. In view of the recent trends towards infinitely scalable computing (Ackley and Small 2014) such a requirement appears unrealistic.

An alternative is to either store the heuristic values in the environments, similar to pheromone-laying ant-colony optimization agents, or use a function approximation as popular in the field of reinforcement learning and game-playing. Early work on heuristic approximation in real-time search encountered anomalies in the agent behavior where deeper lookahead led to worse performance (Bulitko, Levner, and Greiner 2002; Bulitko 2003). Our preliminary work on implementing LRTA* on a Roomba-style robot that writes down its heuristic values on the floor with a dry-board marker (Figure 4) revealed LRTA*'s sensitivity to faulty heuristic read/write operations. Memory access errors become substantially more prominent as the semiconductor industry explores low-voltage near-threshold computing (Dreslinski et al. 2010). They are also important in *ad hoc* wireless sensor networks where LRTA*-style algorithms have been previously applied (Bulitko and Lee 2006).

[‡]This is possibly due to the fact that wbLRTA*'s parameters w and b were optimized for the lowest suboptimality, ignoring the corresponding scrubbing performance.

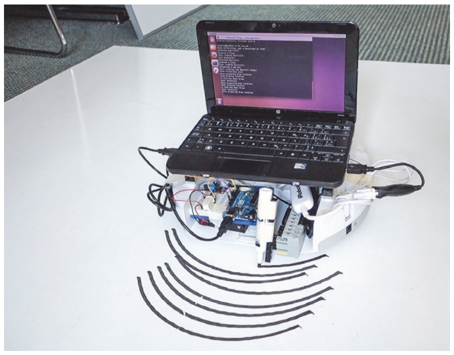

Figure 4: An *iRobot* implements the LRTA* algorithm by storing its heuristic values in the environment. It writes them down on the floor as a sequence of round strokes with a whiteboard marker and reads them back with a down-facing dark/light sensor. In the image the agent is writing down the heuristic value of 8 for its current state.

These findings call for an investigation on how robust real-time heuristic search algorithms are to computational faults. We took a step in that direction by measuring how robust real-time heuristic search algorithms are to access (i.e., input/output) errors in their heuristic function. Note that such errors are different from the inherent error of a heuristic function relative to the perfect heuristic (i.e., $h \neq h^*$). Specifically, we approximate access errors by adding Gaussian noise to each heuristic value read/written from/to the memory. We fixed the mean of the Gaussian distribution at zero and used the standard deviation $\sigma \in \{1, 5, 10\}$. To keep the experiments tractable, we upper-bounded suboptimality at 50 by terminating an agent as soon as its travel cost on a problem exceeds 50 times its optimal solution cost. We then measured the percentage of problems that the algorithm was able to solve not exceeding the quota. The results for 10000 random problems are found in Table 2.

Robustness often requires redundancy and hence algorithms can trade efficiency for robustness (Ackley 2013). Interestingly, this does not appear to be the case among the algorithms considered here: it appears that wbLRTA* yields the best suboptimality and is also most robust to the noise. It is possible that wbLRTA* is more robust to the particular type of heuristic access errors than other algorithms because (i) it takes heuristic values of several neighbors to update the heuristic of the current state and (ii) it raises the heuris-

Table 2: Problems solved (%) with suboptimality $\alpha \leq 50$.

Algorithm	$\sigma = 1$	$\sigma = 5$	$\sigma = 10$
LRTA*	49%	0%	0%
RTA*	58%	50%	36%
aLRTA*	49%	0%	0%
daLRTA*	26%	9%	7%
wLRTA*	81%	77%	64%
wdaLRTA*	8%	4%	11%
daLRTA*+E	7%	11%	10%
wbLRTA*	83%	78%	70%

tic values aggressively, thereby creating larger differences between heuristic values of the neighbors.

To illustrate, suppose that the state on the other end of the dashed edge in Figure 1 has $h = 10.5$. Without the noise, the agent in state s_Z would exit the depression since $11 > 10.5$ (left plot) and $15 > 10.5$ (right plot). However, when the noise is present, the heuristic value of 11 can be misread as, say, $11 - 1.5 = 9.5$ which would cause the agent to stay in the depression as $9.5 < 10.5$. Larger differences between heuristic values are more robust: $15 - 1.5 = 13.5$ is still greater than 10.5 and the agent will still exit the depression. This reasoning is supported by the fact that the second best algorithm, wLRTA*, also uses a very aggressive learning ($w = 128$). wbLRTA* can be more aggressive than wLRTA* for the same w: there is a little chance that the heuristic value of 55 (right plot of Figure 2) would drop below 10.5 even with the Gaussian noise of $\sigma = 10$. Future work will examine robustness of wbLRTA* to other types of heuristic access errors.

7 Current Limitations and Future Work

While the simple combination of weighting the partial average of the f values appears to yield better first-trial solutions, we do yet know to what extent this combination will improve performance of more complex algorithms such as LSS-LRTA* (Koenig and Sun 2009) or f-LRTA* (Sturtevant and Bulitko 2011). Also, other types of computational faults may be considered in the robustness experiments. For instance, an agent may think it is in a different state than it is actually in. Thus it may write down the heuristic value for a wrong state. One can also simulate memory decay over time by increasing the read noise for older heuristic values (cf. pheromone evaporation in ant-colony algorithms).

8 Conclusions

We compared several heuristic learning rules in the stripped-down settings of a real-time heuristic search agent. In a large-scale empirical evaluation on the standard benchmark set, we showed that weighted averaging f over a partial neighborhood produces state-of-the-art results. Our new algorithm is parameterized, generalizes some previous work and is robust to heuristic noise. We theoretically proved it to be complete for all valid parameter combinations.

Acknowledgments

This research has been supported by the National Science and Engineering Research Council. Seth Traverse and Frederic Sauve-Hoover worked on the physical implementation of LRTA* on the *iRobot* platform.

9 Proofs

Lemma 2 Whenever an agent travels from state s_t to state s_{t+1}, the arrival heuristic of the latter state is at least ϵ less than the departing heuristic of the former state. Here $\epsilon > 0$ does not depend on the state.

Proof. Whenever the agent moves from a state s_t to a state s_{t+1}, according to lines 5 and 7 in Algorithm 2 these inequalities hold:

$$\begin{aligned}
h_{t+1}(s_t) &\geq \max\left\{ h_t(s_t),\ w \cdot \operatorname*{avg}_{s \in N_b^f(s_t)} (c(s_t, s) + h_t(s)) \right\} \\
&\geq w \cdot \operatorname*{avg}_{s \in N_b^f(s_t)} (c(s_t, s) + h_t(s)) && (8) \\
&\geq \operatorname*{avg}_{s \in N_b^f(s_t)} (c(s_t, s) + h_t(s)) && (9) \\
&\geq \min_{s \in N(s_t)} (c(s_t, s) + h_t(s)) && (10) \\
&= c(s_t, s_{t+1}) + h_t(s_{t+1}) && (11) \\
&\geq h_t(s_{t+1}) + \epsilon && (12)
\end{aligned}$$

where $\epsilon = \min_{(x,y) \in E} c(x, y)$. Since $|E| < \infty$ and all edge costs are strictly positive, the minimum is well defined and is positive itself. □

Lemma 3 Whenever an agent moves from a state s_t to a state s_{t+1} the heuristic value of state s_t is either unchanged or raised by at least μ.

Proof. First, due to the $\max$ operator in line 5 of Algorithms 1 and 2, any change to the heuristic (learning) implies raising the heuristic. According to line 7, the amount of raise is lower-bounded by μ. □

Lemma 4 For any $b \in [0, 1]$, $w \in [1, \infty)$, wbLRTA* will unboundedly raise heuristic values of all states in any $S' \subset S$ if (i) the agent is only visiting the states in S' after some initial time t':

$$\forall t > t' \, [s_t \in S'] \tag{13}$$

and (ii) each state in S' is visited infinitely many times:

$$\forall s' \in S' \, [|\{t \mid s_t = s'\}| = \infty] \,. \tag{14}$$

Proof. Part I. We will first show that state re-visitation implies learning (i.e., raising the heuristic). Suppose that is not the case and no learning happened during re-visitation. Consider a sequence of states $X = (s_t, \ldots, s_{t+k})$ such that all the states in the sequence except its ends are distinct. The end states of X are the same: $s_t = s_{t+k}$, $k > 1$ and $X \subset S'$. As we assumed that no learning took place while the agent traversed X between the times t and $t + k$, we have:

$$\forall s \in X \, \forall t', t'' \in \{t, \ldots, t + k\} \; [h_{t'}(s) = h_{t''}(s)] \tag{15}$$

which allows us to omit the time index and denote $h_{t'}, t \leq t' \leq t + k$, by just h.

From Lemma 2 we derive $h(s_{t'}) > h(s_{t'+1})$ for any $t' \in \{t, \ldots, t+k-1\}$. In other words, while traversing X the heuristic values are monotonically decreasing. Therefore, the heuristic value of the last state in X is strictly below the heuristic value of the first state in X but they are the same state which makes it impossible. This means that the agent cannot revisit states without raising heuristic values (i.e., learning) somewhere along the way. Since each heuristic raise is lower bounded by μ (Lemma 3) and the number of re-visits to states in S' is unbounded (14), the heuristic of at least some states in S' will rise unboundedly ($\star$).

Part II. We will now show that heuristic values of *all* states in S' will be raised unboundedly. Suppose that is not the case. Then there is a non-empty subset $S'' \subset S'$ comprised of all states that despite being re-visited infinitely many times (14), have their h values upper bounded:

$$S'' = \{s' \in S' \mid \exists u_{s'} < \infty \forall t \, [h_t(s') \le u_{s'}]\}. \tag{16}$$

Since $|S''| \le |S| < \infty$, there exists a single finite upper bound for all of them:

$$\exists u < \infty \forall s' \in S'' \forall t \, [h_t(s') \le u]. \tag{17}$$

Since each update raises the heuristic by at least μ, there must exist a time t' after which the agent will not raise the heuristic of any state in S'', despite continuing to visit them ($\star\star$). Furthermore, the set S'' includes at least two connected states which are immediate neighbors (i.e., connected by a single edge). Indeed, whenever the state $s'' \in S''$ is visited by the agent, its heuristic value is updated in such a way that:

$$h_{t+1}(s'') \ge \max\left\{ h_t(s''),\ w \cdot \operatorname*{avg}_{s \in N_b^f(s'')} (c(s'', s) + h_t(s)) \right\}. \tag{18}$$

Since $h_{t+1}(s'')$ remains upper-bounded by u for any t, it follows that: $\forall t \left[w \cdot \operatorname*{avg}_{s \in N_b^f(s'')} (c(s'', s) + h_t(s)) \le u \right]$ which implies $\forall t \left[\min_{s \in N(s'')} (c(s'', s) + h_t(s)) \le u \right]$. Therefore, the state s'' has an immediate neighbor s° whose heuristic is upper bounded by $u - \min_{(x,y)\in E} c(x,y)$. Thus s° also belongs to S'' (Figure 5).

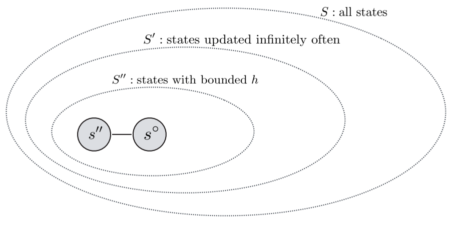
S : all states
S' : states updated infinitely often
S'' : states with bounded h
s''
s°

Figure 5: To the proof of Lemma 4.

Now consider the set $S' \setminus S''$ comprised of the states that are visited indefinitely (14) and have their heuristic values rise unboundedly. At least one such state exists, according to Part I of the proof ($\star$). Once the heuristic values of all states in $S' \setminus S''$ rise sufficiently above u and after time t', the agent will necessarily execute a cycle over the states contained entirely in S'' (such a cycle is possible over, for instance, s'' and s°). This is so because whenever the agent is in any state in S'' whose immediate neighbors are both in S'' and $S' \setminus S''$, it would necessarily go to a neighbor in S'' (whose heuristic is upper-bounded) and not to a neighbor in $S' \setminus S''$ (whose heuristic can will be arbitrarily high at that moment). Thus, by Part I of the proof, the agent will raise the heuristic of at least one state on this cycle. This means that the heuristic of a state in S'' will be raised after time t' — a contradiction to ($\star\star$). We have now shown that unlimited revisitation of states in S' leads to unbounded rise of the heuristic values in all states in S'. $\square$

Theorem 1 wbLRTA* is complete for any beam width $b \in [0, 1]$ and any weight $w \in [1, \infty)$.

Proof. We will prove that for any search problem $\mathbf{S} = (S, E, c, s_0, s_g, h)$ the agent using Algorithm 2 is guaranteed to terminate in the goal state s_g. The proof is by contradiction. Suppose there exists a search problem $\mathbf{S}$ and the control parameters w and b such that the agent, guided by wbLRTA*, is incomplete on the problem. Since the agent can terminate only in the goal state (line 3 in Algorithm 2), incompleteness means that the agent never terminates. Since the set of states S is finite there is necessarily a subset of states $S' \subset S$ that satisfies conditions (13) and (14) from Lemma 4. In other words, after time t' the agent will visit only states in S' and visit each indefinitely many times.

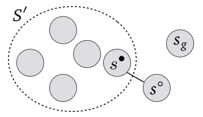

Figure 6: To the proof of Theorem 1.

Consider frontier of S': $\partial S' = \{s \in S' \mid \exists s^\circ \in S \setminus S' \ \& \ (s, s^\circ) \in E\}$. Since the goal state $s_g \notin S'$ (for otherwise the agent would terminate upon its first arrival in s_g) and the graph is connected the frontier is not empty: $\partial S' \neq \emptyset$. Figure 6 shows one such frontier state $s^\bullet \in \partial S'$. By Lemma 4, the agent will increase the heuristic values of all states in $\partial S'$ indefinitely. Thus, at some point t, the $c + h$ values of all S'-contained neighbors of any state $s^\bullet \in \partial S'$ will necessarily exceed any $c + h$ value of its $S \setminus S'$-contained neighbors:

$$\exists t \forall s^\bullet \in \partial S' \left[\min_{s \in N(s^\bullet) \cap S'} (c(s^\bullet, s) + h_t(s)) \right. \tag{19}$$

$$\left. > \max_{s^\circ \in N(s^\bullet) \setminus S'} c(s^\bullet, s^\circ) + h_t(s^\circ) \right]. \tag{20}$$

Then line 4 in Algorithm 2 will cause the agent to take edge $(s^\bullet, s^\circ)$ thereby leaving S' which contradicts the proposition that the agent will be contained in S' which was derived from the assumption that the agent is incomplete on $\mathbf{S}$. $\square$

References

Ackley, D. H., and Small, T. R. 2014. Indefinitely scalable computing = artificial life engineering. In *ALIFE 14*, 606–613.

Ackley, D. H. 2013. Beyond efficiency. *Communications of the ACM* 56(10):38–40.

Björnsson, Y.; Bulitko, V.; and Sturtevant, N. 2009. TBA*: Time-bounded A*. In *IJCAI*, 431–436.

Botea, A. 2011. Ultra-fast optimal pathfinding without runtime search. In *AIIDE*, 122–127.

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

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

Bulitko, V.; Luštrek, M.; Schaeffer, J.; Björnsson, Y.; and Sigmundarson, S. 2008. Dynamic control in real-time heuristic search. *JAIR* 32:419–452.

Bulitko, V.; Björnsson, Y.; and Lawrence, R. 2010. Case-based subgoaling in real-time heuristic search for video game pathfinding. *JAIR* 39:269–300.

Bulitko, V.; Levner, I.; and Greiner, R. 2002. Real-time lookahead control policies. In *AAAI/KDD/UAI Workshop on Real-Time Decision Support and Diagnosis Systems*.

Bulitko, V.; Rayner, D. C.; and Lawrence, R. 2012. On case base formation in real-time heuristic search. In *AIIDE*, 106–111.

Bulitko, V. 2003. Lookahead pathologies and meta-level control in real-time heuristic search. In *Euromicro Conference on Real-Time Systems*, 13–16.

Bulitko, V. 2004. Learning for adaptive real-time search. *CoRR* cs.AI/0407016.

Bulitko, V. 2016a. Evolving real-time heuristic search algorithms. In *ALIFEXV*, (in press).

Bulitko, V. 2016b. Searching for real-time search algorithms. In *SoCS*, (in press).

Dreslinski, R. G.; Wieckowski, M.; Blaauw, D.; Sylvester, D.; and Mudge, T. 2010. Near-threshold computing: Reclaiming Moore's law through energy efficient integrated circuits. *IEEE* 98(2):253–266.

Felner, A.; Zahavi, U.; Holte, R.; Schaeffer, J.; Sturtevant, N.; and Zhang, Z. 2011. Inconsistent heuristics in theory and practice. *AI* 175(9):1570–1603.

Furcy, D., and Koenig, S. 2005. Limited discrepancy beam search. In *IJCAI*, 125–131.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. on Systems Science and Cybernetics* 4(2):100–107.

Hernández, C., and Baier, J. A. 2012. Avoiding and escaping depressions in real-time heuristic search. *JAIR* 43:523–570.

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

Huntley, D. A., and Bulitko, V. 2013. Search-space characterization for real-time heuristic search. *CoRR* abs/1308.3309.

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

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

Koenig, S. 2001. Agent-centered search. *Artificial Intelligence Magazine* 22(4):109–132.

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

Lawrence, R., and Bulitko, V. 2013. Database-driven real-time heuristic search in video-game pathfinding. *IEEE Trans. Comput. Intellig. and AI in Games* 5(3):227–241.

Rayner, D. C.; Davison, K.; Bulitko, V.; Anderson, K.; and Lu, J. 2007. Real-time heuristic search with a priority queue. In *IJCAI*, 2372–2377.

Rivera, N.; Baier, J. A.; and Hernández, C. 2015. Incorporating weights into real-time heuristic search. *AI* 225:1–23.

Sharon, G.; Sturtevant, N. R.; and Felner, A. 2013. Online detection of dead states in real-time agent-centered search. In *SoCS*, 167–174.

Shimbo, M., and Ishida, T. 2000. Towards real-time search with inadmissible heuristics. In *ECAI*, 609–613.

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

Shue, L.-Y., and Zamani, R. 1993a. An admissible heuristic search algorithm. In *ISMIS*, volume 689 of *LNAI*, 69–75.

Shue, L.-Y., and Zamani, R. 1993b. A heuristic search algorithm with learning capability. In *ACME Transactions*, 233–236.

Shue, L.-Y.; Li, S.-T.; and Zamani, R. 2001. An intelligent heuristic algorithm for project scheduling problems. In *Annual Meeting of the Decision Sciences Institute*.

Sturtevant, N. R., and Bulitko, V. 2011. Learning where you are going and from whence you came: H- and G-cost learning in real-time heuristic search. In *IJCAI*, 365–370.

Sturtevant, N. R., and Bulitko, V. 2014. Reaching the goal in real-time heuristic search: Scrubbing behavior is unavoidable. In *SoCS*, 166–174.

Sturtevant, N., and Buro, M. 2005. Partial pathfinding using map abstraction and refinement. In *AAAI*, 1392–1397.

Sturtevant, N. R.; Bulitko, V.; and Björnsson, Y. 2010. On learning in agent-centered search. In *AAMAS*, 333–340.

Sturtevant, N. 2007. Memory-efficient abstractions for pathfinding. In *AIIDE*, 31–36.

Sturtevant, N. R. 2012. Benchmarks for grid-based pathfinding. *IEEE TCIAIG* 4(2):144 – 148.