Proceedings, The Thirteenth AAAI Conference on
Artificial Intelligence and Interactive Digital Entertainment (AIIDE-17)

Robustness of Real-Time Heuristic Search Algorithms to Read/Write Error in Externally Stored Heuristics

Mina Abdi Oskouie, Vadim Bulitko
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
{abdiosko|bulitko}@ualberta.ca

Abstract

Real-time heuristic search algorithms follow the agent-centered search paradigm wherein the agent has access only to information local to the agent's current position in the environment. This allows agents with constant-bounded computational faculties (e.g., memory) to take on search problems of progressively increasing sizes. As the agent's memory does not scale with the size of the search problem, the heuristic must necessarily be stored externally, in the environment. Storing the heuristic in the environment brings the extra challenge of read/write errors. In video games, introducing error artificially to the heuristics can make the non-player characters (NPC) behave more naturally. In this paper, we evaluate effects of such errors on real-time heuristic search algorithms. In particular, we empirically study the effects of heuristic read redundancy on algorithm performance and compare its effects to the existing technique of using weights in heuristic learning. Finally, we evaluate a recently proposed technique of correcting the heuristic with a one-step error term in the presence of read/write error.

1 Introduction

Real-time heuristic search follows the agent-centered framework of Koenig (2001). Such an agent occupies a single vertex in the search graph which it changes by traversing the graph's edges. Being agent-centered, it can only use information around its current vertex when deciding on the edge to traverse. The decision can also be influenced by the assumptions that it makes about other parts of the terrain. Such locality of heuristic access is desirable when the agent's cognitive ability does not scale with the size of the problem it is solving (e.g., in *ad hoc* wireless networks (Bulitko and Lee 2006) or, more generally, any infinitely scalable computing (Ackley and Small 2014)) and/or when locality of memory access is desirable (e.g., when using disks to store information about the search space (Korf and Schultze 2005) or taking advantage of CPU cache coherence).

Heuristic search algorithms normally assume that heuristic values can be stored perfectly and thus any heuristic error is due to the simplifications/abstractions taken in defining the heuristic function (e.g., ignoring map obstacles in the Manhattan distance heuristic). While robotics show heuristics can become noisy in the procedure of storing whether internally, in the memory of robots, or externally, in the environment. Lelis et al. introduced a scheme to correct the errors caused by the memory corruption (Lelis et al. 2016) [1]. They assumed that the corruption can flip bits of the heuristics stored in the memory and make them inadmissible.

Storing heuristic functions in the environment, is inspired by ants who lay pheromone on the ground for path finding. The intensity of the pheromone can be affected by natural phenomena like temperature or rain and eventually mislead the ant (Van Oudenhove et al. 2012). Recent real-time heuristic search literature also observed the same errors (read/write errors) when implementing LRTA* (Korf 1990) on a *Roomba*-style robot (Traverse and Suave-Hoover 2014). The robot stored its heuristic function externally by writing it with a blackboard marker on the floor. Doing so led to read/write errors in the heuristic. Bulitko and Sampley (2016) studied the effects of these errors empirically in a simulation. Specifically, they modeled the heuristic read/write errors as Gaussian noise added to the true heuristic value. They compared eight heuristic search algorithms in the presence of noise and showed that wLRTA* (previously suggested by Rivera, Baier, and Hernández (2015)) and their own wbLRTA* were most robust to the heuristic read/write errors. They speculated that the robustness may be due to the more aggressive updates due to weight and/or due to the learning operator that averages heuristic information around the current state as opposed to the LRTA*'s traditional minimum operator.

Humans and animals do not always find the perfect path. Their mistakes are due to wrong estimations or assumptions they have in their mind. These phenomena can be modeled by corrupted heuristics. Adding errors to heuristics can make the NPC behave less robotic and more like humans or animals. As the information gets more inaccurate (i.e. mishearing the conversation from longer distances), the errors become more intense. Humans behave differently given inaccurate information. Having a method to alleviate the error

[1] Disk read/write errors in fully searching the 15 puzzle (Korf and Schultze 2005) were eventually solved by using error-correcting disk storage (RAID). Furthermore, even solid-state memory can exhibit read/write errors especially in low-voltage implementations (Dreslinski et al. 2010).

partially, can help us capture these differences in NPCs.

In this paper, we build on their work and make the following contributions. First, we attempt to correct the read heuristic error by retrieving the heuristic multiple times for a single state. We compare the effects of this idea to that of introducing weights to the learning rule of LRTA* (i.e., wLRTA*). Second, we evaluate the recent heuristic error-correction technique of O'Ceallaigh and Ruml (2015) in the presence of the read/write heuristic noise. We show that the error correction based on averaging one-step heuristic error is prone to systematic bias in the presence of read/write heuristic errors. We then show that the multiple retrievals technique does help but not enough to recommend using their error correction technique over the regular LRTA*.

2 Problem Formulation

For continuity we use the same notation as the work we extend (Bulitko and Sampley 2016) and reproduce it here for the reader's convenience. The *search problem* is defined as the tuple $(S, E, c, s_{\text{start}}, s_{\text{goal}}, h)$ where S is a finite set of *states* and $E \in S \times S$ is a set of *edges* between the states. To prove the completeness of real-time heuristic search algorithms, S is commonly assumed to be finite. Although we do not consider the completeness here, we assume it finite to be in line with previous endeavors. The search graph is defined by S and E and is assumed undirected: $\forall a, b \in S\,[(a, b) \in E \implies (b, a) \in E]$ and lacking self-loops: $\forall s \in S\,[(s, s) \notin E]$. Each edge in E has a strictly positive weight: $c : E \to \mathbb{R}^+$. The edge weights remain unchanged during search (i.e., the search graph is stationary). The states a and b are *immediate neighbors* iff there exists an edge between them: $(a, b) \in E$. The set of all immediate neighbors of a, called *neighborhood* of a, is denoted by $N(a)$. A path P is a sequence of states $(s_0, s_1, \dots, s_n)$ in which for each $i \in \{1, \dots, n\}$, s_{i-1} and s_i are immediate neighbors. We assume that the graph is safely explorable and the goal state is reachable from any state which an agent can get to from the start state.

The agent is first put in the start state, s_{start}. It moves toward the goal state, s_{goal}, by traversing edges in the graph. The state that agent occupies at time t is denoted as s_t . The problem is solved at earliest time T, when $s_T = s_{\text{goal}}$. The *solution* is a path $P = \{s_{\text{start}}, \dots, s_{\text{goal}}\}$ that agent has traversed in order to reach the goal states. The *solution cost* is defined as the cumulative cost of all edges in the solution path P: $C(P) = \sum_{t=0}^{T-1} c(s_t, s_{t+1})$. The cost of the shortest path between two arbitrary states a and b is denoted as $h^*(a, b)$. The *suboptimality* (α) of the agent using algorithm A is the ratio of the cost of the solution P found by this agent to the cost of the shortest possible path $\alpha(A) = {C(P)}/{h^*(s_{\text{start}}, s_{\text{goal}})}$.

The *heuristic function* $h : S \to [0, \infty)$ estimates the distance from each state to the goal. The agent can access and update hs any time. The heuristic of the goal state is always zero. An externally stored heuristic may have *read/write errors*. Specifically, an agent attempting to retrieve a heuristic value for a state may get a different reading on each attempt. In this paper we follow the model of Bulitko and Sampley (2016) [2] and model the value read as $h(s) + x$ where x is randomly sampled from the normal distribution $N(0, \sigma)$ with the mean $\mu = 0$ and the standard deviation σ. Likewise, when the agent writes out a new value of the heuristic for state s, the actual value written out is $h(s) + x$ with $x \sim N(0, \sigma)$. In this paper, we focus on *real-time* heuristic search algorithms whose planning time per move is upper bounded by a constant independent of the total number of states in the search graph. This excludes classical algorithms such as A* (Hart, Nilsson, and Raphael 1968) but includes a large number of modern real-time algorithms (Bulitko 2016). In-line with the literature (Bulitko and Lee 2006) we will evaluate the algorithms by their solution suboptimality as defined above. Additionally, as a total measure of computational effort spent to derive a solution, we measure the *goal achievement input/output operations* (GAIO) which is the total number of heuristic read and write operations performed by the agent before it reaches the goal. GAIO is inspired by the goal-achievement time (GAT) of Hernández et al.; Burns, Kiesel, and Ruml (2012; 2013) but avoids having to worry about the amount of time an agent spends traversing an edge as well as makes the measurement more reliable (since time at the scale of micro-/milli-seconds is subject to measurement errors, especially when running in a multi-threaded environment). Within our algorithms, the majority of planning time per move is proportional to heuristic input/output operations (as detailed in the next section), allowing GAIO to reliably capture the cumulative per-problem computational effort of different algorithms.

3 Framework

In this work, we evaluate two algorithms. We use the fixed lookahead of 1 (i.e., consider only the agent's immediate neighbors) as such myopic algorithms form the base for more sophisticated/powerful algorithms while being simpler to present and discuss. First, we consider weighted LRTA* adapted from work of Bulitko and Sampley; Rivera, Baier, and Hernández (2016; 2015).

Algorithm 1: Simple Weighted LRTA*

```
input : search problem $(S, E, c, s_{\text{start}}, s_{\text{goal}}, h, n)$, weight $w$
output: solution path $(s_{\text{start}}, s_1, \dots, s_{\text{goal}})$
1 $t \leftarrow 0$
2 $s_t \leftarrow s_{\text{start}}$
3 $h_t \leftarrow h$
4 while $s_t \neq s_{goal}$ do
5     $s_{t+1} \leftarrow \arg\min_{s \in N(s_t)} [c(s_t, s) + h_t(s)]$
6     $h_{t+1}(s_t) \leftarrow \max\{h_t(s_t), \min_{s \in N(s_t)} [w \cdot c(s_t, s) + h_t(s)]\}$
7     $t \leftarrow t + 1$
```

In the weighted LRTA* (Algorithm 1) the agent interleaves planning and learning phases. As long as the goal is not reached (line 4), the agent selects the next state as its im-

[2] Earlier Piltaver, Luštrek, and Gams; Sadikov and Bratko (2012; 2006) modeled heuristic errors by Gaussian distribution.

mediate neighbor with the minimum $f = c+h$ cost (line 5).[3] It then updates its heuristic in the current state using the weighted update rule in line 6. The maximum guarantees that heuristics never decrease in the updating procedure. Doing so would not be necessary if we had assumed that the heuristics are consistent. We do not assume admissibility or consistency of the heuristic and thus use the maximum explicitly. This is in line with wLRTA* evaluated by Bulitko and Sampley (2016).

The second algorithm we evaluate is a simplified version of the Dynamic $\hat{f}$ algorithm (O'Ceallaigh and Ruml 2015). In our version (Algorithm 2) the agent always used the lookahead of 1 by considering only the immediate neighbors of the current state. The resulting algorithm is largely identical to the weighted LRTA* presented above with the following differences. First, no weight is used in learning. Second, in line with the original Dynamic $\hat{f}$ algorithm by O'Ceallaigh and Ruml (2015), Algorithm 2 uses $\hat{f}$ instead of wLRTA*'s f in choosing its next state. Specifically, in line 11, the next state s_{t+1} is selected as the neighbor with the lowest $\hat{f}$ cost where $\hat{f}$ is $f = c + h$ cost with an error-correcting add-on $\bar{\epsilon} \cdot d(\text{support}(s))$. The intuition is that each edge remaining to be traversed by the agent until the goal state adds a heuristic error which the agent needs to correct for in order to make a more informed move. The number of edges remaining until the goal state is unknown and is estimated by the function $d : S \to \mathbb{N}$ supplied to the agent.[4] The per-edge error $\bar{\epsilon}$ is the average of per-edge f errors seen so far (line 10). Each such per-edge f error is computed in line 9 as the difference of the lowest f among the current state's neighbors (i.e., $\min_{s \in N(s_t)}[c(s_t, s) + h(s_t)]$) and the current state's own f (i.e., $h_t(s_t)$).

Algorithm 2: Simplified Dynamic $\hat{f}$

```
   input : search problem $(S, E, c, s_{\text{start}}, s_{\text{goal}}, h, n)$, heuristic $d$
   output: solution $(s_{\text{start}}, s_1, \dots, s_{\text{goal}})$
1  $t \leftarrow 0$
2  $s_t \leftarrow s_{\text{start}}$
3  $h_t \leftarrow h$
4  $\forall s \in S\,[\text{support}(s) \leftarrow s]$
5  $\epsilon \leftarrow 0$
6  while $s_t \neq s_{goal}$ do
7      $h_{t+1}(s_t) \leftarrow \max\{h_t(s_t), \min_{s \in N(s_t)}[c(s_t, s) + h_t(s)]\}$
8      $\text{support}(s_t) \leftarrow \arg\min_{s \in N(s_t)}[c(s_t, s) + h_t(s)]$
9      $\epsilon \leftarrow \epsilon - h_t(s_t) + \min_{s \in N(s_t)}[c(s_t, s) + h_t(s)]$
10     $\bar{\epsilon} \leftarrow \epsilon/t+1$
11     $s_{t+1} \leftarrow \arg\min_{s \in N(s_t)}[c(s_t, s) + h_t(s) + \bar{\epsilon} \cdot d(\text{support}(s))]$
12     $t \leftarrow t + 1$
```

4 Related Work

Our work builds on the study of real-time search performance in the presence of read/write error noise (Bulitko and Sampley 2016). That work was itself inspired by the read/write errors observed by Traverse and Suave-Hoover (2014) while implementing LRTA* on a *Roomba*-style robot which stored the heuristic externally in the environment. Bulitko and Sampley (2016) reported the percentage of search problems solved by eight algorithms under a prior upper bound on the solution suboptimality. The eight algorithms considered were a mixture of the classic algorithms LRTA*, RTA* (Korf 1990) and more contemporary ones: aLRTA*, daLRTA*, wLRTA*, wdaLRTA*, daLRTA*+E and wbLRTA*. These algorithms used the weighting in the learning rule (Rivera, Baier, and Hernández 2015), tracking previously visited states (Hernández and Baier 2012), pruning expendable states (Sharon, Sturtevant, and Felner 2013) and lateral learning (Bulitko and Sampley 2016). While interesting and pioneering, Bulitko and Sampley (2016)'s work neither explicitly measured the resulting solution suboptimality nor considered the total computational effort.

Ruml's group has been investigating correcting heuristic errors and building more accurate heuristics over the last several years. Thayer and Ruml (2009) considered using distance-to-go (i.e., the number of edges remaining to traverse, regardless of their cost) in a non-real-time search setting. Thayer, Dionne, and Ruml (2011) introduced the idea of measuring one-step heuristic errors and using them in a correction term to obtain better heuristics. More recently, O'Ceallaigh and Ruml (2015) combined these techniques in a new real-time heuristic search algorithm that corrects the heuristic error using the average one-step heuristic error observed by the agent as well as deciding on the amount of planning on each move (meta-reasoning). Following the common practice in the field, they assumed that the agent can store the heuristic perfectly and therefore the only errors in the heuristic are due to the simplification/abstraction used in its construction.

While none of the existing work fully addresses the problem we tackle in this paper, our paper builds on the research of the last two years (O'Ceallaigh and Ruml 2015; Bulitko and Sampley 2016).

5 Multiple Heuristic Retrievals

In this paper, we tackle the read/write heuristic errors with averaging redundant heuristic read operations. In other words, whenever the agent needs to look up the heuristic value of a state, it does so R times and then averages the results. This simple idea trades heuristic accuracy for running time and is a simple case of trading efficiency for robustness (Ackley 2013). The sample mean is an unbiased estimator of the actual heuristic value stored in the environment. The higher the R the more precise the average of the multiple heuristic readings will be but the longer the search will take. Algorithms with higher R are expected to have lower

[3]Throughout this paper ties among neighbors are broken in a fixed order (e.g., in grid pathfinding we may prefer neighbors to the north to neighbors to the west).

[4]As O'Ceallaigh and Ruml (2015), we compute d with respect to the state's support defined as its neighbor whose f value was used to compute the state's current h (line 7). Initially each state's support is the state itself (line 4). The support is updated in line 8.

suboptimality but longer overall planning time (GAIO).[5]

Note that the multiple retrieval technique does not address write errors in the heuristic. Indeed, an agent attempting to store $h(s)$ is actually storing $h(s) + x, x \sim N(0, \sigma)$ an estimate of which will later be read. Addressing write errors is a subject of future work.

6 Empirical Evaluation

We will first introduce our experimental testbed and then present results of several experiments designed to address previously introduced questions.

6.1 Experimental Testbed

We used 205 different maps from the Moving AI map set (Sturtevant 2012) that were sampled from the following games: *Dragon Age: Origins*, *Starcraft*, *Warcraft III* and *Baldurs Gate II* . As the common practice, the maps were converted to 8-connected two-dimensional grids. Each state was connected to at most 8 neighbors. Cardinal edges had a cost of 1 while diagonal edges cost $\sqrt{2}$. The initial heuristic h between two states $a = (a_x, a_y)$ and $b = (b_x, b_y)$ is the usual octile distance: $h(a, b) = \sqrt{2} \cdot \min\{\Delta_x, \Delta_y\} + \max\{\Delta_x, \Delta_y\} - \min\{\Delta_x, \Delta_y\}$ where $\Delta_x = |a_x - b_x|$ and $\Delta_y = |a_y - b_y|$. The number-of-edges estimate d used in Algorithm 2 is computed similarly except both diagonal and cardinal edges are counted as 1: $d(a, b) = \min\{\Delta_x, \Delta_y\} + \max\{\Delta_x, \Delta_y\} - \min\{\Delta_x, \Delta_y\} = \max\{\Delta_x, \Delta_y\}$.

To have meaningful results and to make sure the experiments could run in a practical amount of time we generated a problem set as follows. On each of the 205 maps, we generated problems whose start and end states were chosen randomly. Out of such, we picked 5 problems per map whose LRTA* suboptimality was in $(10, 70)$ (in the absence of read/write heuristic errors). We also made sure that each problem could be solved by the basic LRTA* even in the presence of read/write heuristic errors ($\sigma \in \{5, 10, 15\}$) and that the resulting suboptimality did not exceed 200 on the first attempt.[6] This process resulted in a set of 1025 problems which we then used for all our experiments.

Each heuristic value h read in or written out by the algorithms had a random value added to it. Such values were sampled from a normal distribution $N(0, \sigma)$ with the standard deviation $\sigma \in \{0, 5, 10, 15\}$. Note that both of our algorithms update/learn only h which is stored externally and thus is subject to read/write noise. The secondary heuristic d is computed procedurally and is never updated. It is not stored in the environment and thus is not subject to read/write errors. The fact that the errors are different in each trial, makes the agent behave differently. To gain a better insight about the algorithms, each of the following experiments are repeated 10 times and the average is reported.

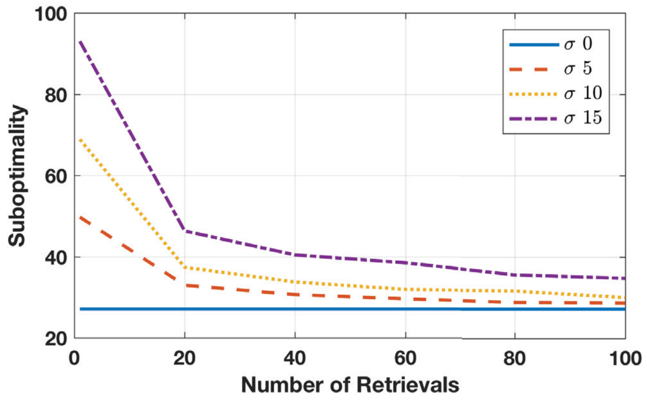

Figure 1: Average suboptimality of LRTA* as a function of the number of heuristic retrievals R and the standard deviation of read/write error σ.

6.2 Effects of Multiple Retrievals on Solution Quality and Total Planning Time

In this section, we explore how averaging over multiple heuristic retrievals influences performance of LRTA*. For this matter we tested LRTA* in the presence of read/write errors with the standard deviation of 0, 5, 10 and 15. Figure 1 show the suboptimality of the algorithm. As we expected, applying multiple retrievals helps LRTA* achieve better suboptimality. More retrievals are needed to combat higher σ. Multiple heuristic retrievals per move helps the agent make more informed moves and achieve shorter solutions. This does not compensate for the larger number of read/write operations per move and the total GAIO monotonically increases with R.

6.3 Multiple Retrievals and Learning Weight

In this section we investigate the influence of applying multi-retrieval technique and using the learning weight w. We ran wLRTA* (Algorithm 1) with the weight $w \in \{1, 5, 10, 15\}$ and the number of multiple heuristic retrievals $R \in \{1, 10, 20, 30, 40, 50, 60, 100\}$. For each of the combinations of w and R we ran wLRTA* on our set of 1025 problems with the read/write heuristic error of $\sigma = 15$. Figures 2 and 3 show suboptimality and GAIO respectively, averaged over all problems in the set as functions of R and w.

Increasing the learning weight and retrieving heuristics both contribute to improving average suboptimality. Using both of these techniques, we can achieve suboptimality below 20 while the suboptimality of the original LRTA* ($w = 1, R = 1$) is about 90. Applying learning weight, multiple retrievals and both can improve the agent's performance on more than 71%, 98% and 99% of problems respectively. As we discussed earlier, more retrievals increases GAIO even though it is decreasing the number of visited states. On the other hand, Figure 3 shows that increasing the learning weight can decrease GAIO. This is because a

[5]One can imagine that reducing suboptimality will also improve the total planning time as the agent reaches the goal faster and thus has fewer moves to plan. We investigate this empirically in Section 6.2.

[6]In the presence of read/write heuristic errors, the search becomes stochastic. Thus the fact that LRTA* was able to solve a problem with read/write errors does not guarantee it will do so again as the read/write errors may be different on a different run.

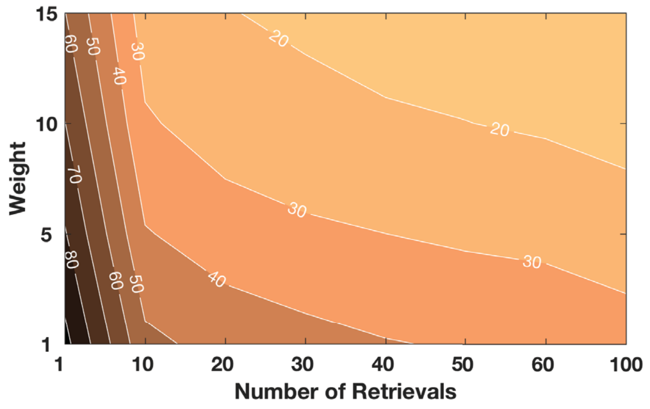

Figure 2: Average suboptimality of wLRTA* as a function of the number of heuristic retrievals R and the learning weight w.

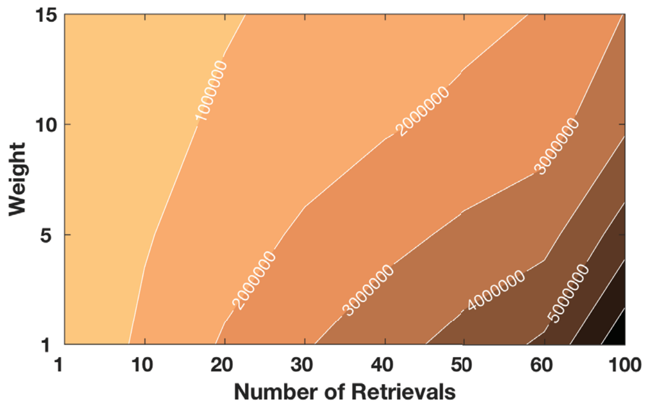

Figure 3: Average GAIO of wLRTA* as a function of the number of heuristic retrievals R and the learning weight w.

higher learning rate improves suboptimality without affecting the planning time per move.

6.4 Simplified Dynamic $\hat{f}$ with Read/Write Errors

In this section we evaluate our simplified dynamic $\hat{f}$ algorithm (Algorithm 2) in the presence of read/write heuristic errors. The algorithm explicitly models one-step errors in f and uses their average to inform its choice of the next state. In doing so it makes several assumptions (O'Ceallaigh and Ruml 2015) which are worth checking in our testbed.

We first examine heuristic error correction term in the absence of read/write heuristic errors. To do so we compare the correction $\bar{\epsilon} \cdot d(\text{support}(s))$ to the actual heuristic error $h^*(s) - h_t(s)$. We compute the quantities for all states s considered by Algorithm 2 in line 7. The left plot in Figure 4 presents the results. For each point the x-coordinate is the heuristic correction of O'Ceallaigh and Ruml (2015) whereas the y-coordinate is the actual heuristic error. If the correction were perfect all points in the scatter plot would land on the 45° line. As the figure shows, they do not. In

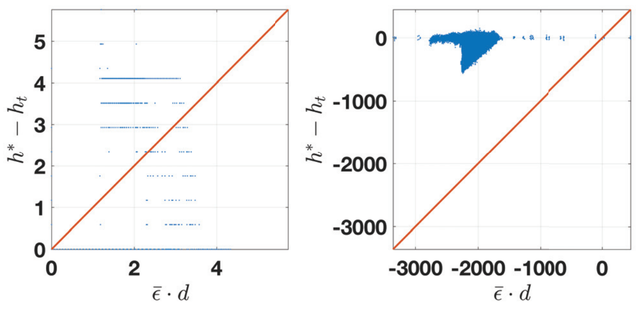

Figure 4: *Left:* all the visited states by LRTA* when it uses $\hat{h}$ to find its next move and there is no read/write error. In this case the agent finds the goal after visiting 1191 states. *Right:* all the visited states by LRTA* when it uses $\hat{h}$ to find its next move and there is read/write error with standard deviation of 15. In this case the agent cannot find the goal after visiting 1132253 states. For both plots R is equal to 1.

particular, there are a number of points on the x axis which indicates states where $h_t = h^*$ but $\bar{\epsilon} \cdot d(\text{support}(s)) > 0$. This happens, for instance, when there are no obstacles between s and s_{goal} which makes the heuristic perfect but does not make $\bar{\epsilon} \cdot d(\text{support}(s)) = 0$.

We now repeat the measurements but this time add normally distributed errors to the read/write operations ($\sigma = 15$). The right plot in Figure 4 shows the results. Here the correcting term $\bar{\epsilon} \cdot d(\text{support}(s))$ is highly negative. The reason lies with the way Algorithm 2 computes per-step error in line 9. In the expression $-h_t(s_t) + \min_{s \in N(s_t)} [c(s_t, s) + h_t(s)]$ used by the algorithm, the min operation chooses neighboring states whose f value (i.e., $c(s_t, s) + h_t(s)$) is particularly low due to the read/write error in the heuristic. While the f value of the current state (i.e., $h_t(s_t)$) is also affected by the read/write error in the heuristic, the chances are that at least one of the multiple neighbors of the current state s_t will have its h_t value read from the environment artificially low. As a result, the lowest f of the neighbors tends to be below the f of the current state and the one-step error ends up being negative. Thus, $\bar{\epsilon}$ will tend to be negative and the correction term $\bar{\epsilon} \cdot d(\text{support}(s))$ will be negative as well.

We confirm this reasoning by plotting the converged value of $\bar{\epsilon}$ in Figure 5. We say that $\bar{\epsilon}$ is converged when the absolute value of difference between its previous value and the new one is less than 0.001. The converged value of $\bar{\epsilon}$ becomes progressively more negative as the standard deviation of the read/write errors σ increases.

The bias brought on by the min operation in estimating ϵ can be reduced by averaging read h over multiple retrievals. As Figure 5 shows, by increasing R from 1 to 750 the final values of $\bar{\epsilon}$ converge. The differences between the values of converged $\bar{\epsilon}$ in 750 retrievals are due to the write error which is not impacted by multiple retrieval technique.

6.5 Pareto-optimality Analysis

We have so far considered two algorithms and two techniques: weighted learning and multi-retrieval heuristics. In

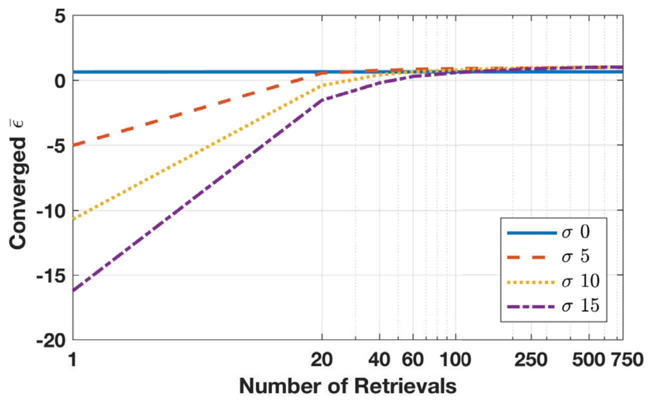

Figure 5: Average of final $\bar{\epsilon}$ over all problems with different number of retrievals and standard deviation of read/write heuristic error.

this section we analyze the resulting variations in the space of our two performance measures: solution suboptimality and the total planning effort (GAIO). We say that an algorithm A is *dominated* by algorithm B iff B is simultaneously better than A with respect to both performance measures. All non-dominated algorithms form a *pareto-optimal frontier*, trading one performance measure for the other.

For this experiment we run wLRTA* and simplified Dynamic $\hat{f}$ algorithm with $w \in \{1, 5, 10, 15\}$ and $R \in \{1, 50, 100, 150\}$. The standard deviation (σ) of read/write error is equal to 15[7]. Note that we excluded $R = 1$ from simplified Dynamic $\hat{f}$ with weight on edge cost in the presence of read/write errors ($\sigma > 0$) as they were unable to solve most problems under the suboptimality cutoff of 10^4 due to the phenomena (large negative $\bar{\epsilon}$) analyzed in Section 6.4.

Each point in Figure 6 represents a parametrized algorithm. The suboptimality and GAIO averaged over our problem set of 1025 problems on 205 maps. Like the previous results, Figure 6 shows that increasing the learning weight not only improves the suboptimality but also reduces the GAIO in both algorithms. We believe this is due to the reason identified by Bulitko and Sampley (2016): a larger weight makes the algorithm update its heuristic more aggressively thereby creating a larger difference between the value of heuristic and its neighbors and discouraging the agent from revisiting states. The explanation is confirmed by the observed decreasing in the average number of visits per state for all settings (e.g. decreases from 6.51 to 1.36 by adding $w = 15$ to LRTA* when there is no read/write error). Averaging over multiple heuristic retrievals can also improve the suboptimality but harms the total planning time (GAIO) as per our analysis in Section 6.2. A practitioner can pick any pareto-optimal algorithm (filled markers) depending on how they would like to trade suboptimality and GAIO.

[7]To apply weighted learning technique on dynamic f, similar to wLRTA*, we only change the learning rule. Specifically, we replace line 7 of algorithm 2 with line 6 in algorithm 1.

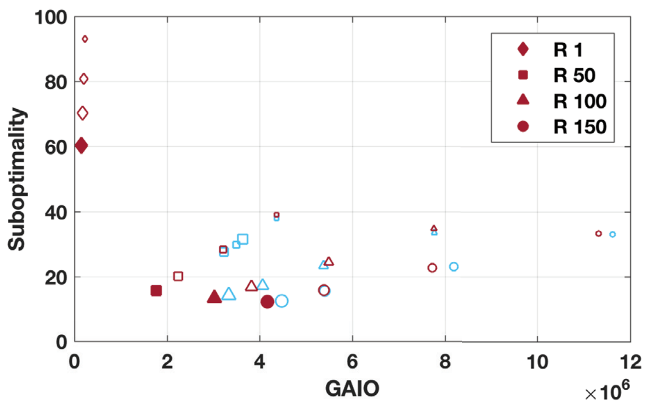

Figure 6: Pareto optimality analysis. Dominated algorithms are shown with a hollow marker. Darker/red markers are for wLRTA*. Lighter/blue markers are for the simplified Dynamic $\hat{f}$ algorithm with the added learning weight w. Both algorithms use averages of multiple heuristic retrievals.

7 Current Limitations and Future Work

This preliminary study opens interesting avenues for future work. First, algorithms with deeper lookahead such as LSS-LRTA* (Koenig and Sun 2009) and the actual Dynamic $\hat{f}$ (O'Ceallaigh and Ruml 2015) need to be investigated in the framework of our study. Second, storing the heuristic externally can additionally lead to cross-talk between heuristic values as well as deterioration of stored values over time. We plan to use biologically (Schultheiss, Cheng, and Reynolds 2015; Merkle, Knaden, and Wehner 2006) and robotically (Svennebring and Koenig 2004) inspired models of such errors. It would also be interesting to replicate our simulation-based study on an actual robot.

8 Conclusions

In this work we explore scalable agent-centered (Koenig 2001) computing and investigate robustness of real-time search algorithms to read/write error in the heuristic stored externally to the agent. This paper builds on the previous work (Bulitko and Sampley 2016) extending it in several important ways. Specifically, we demonstrated the effectiveness of redundant heuristic read operations with and without learning weights. We also demonstrated issues with using average one-step h error in the presence of read/write heuristic errors and how redundant heuristic reads can remedy the issue. Using single-step error, $\bar{\epsilon}$, cannot improve LRTA* even when there is no read/write error. This is not in contrast with the work (O'Ceallaigh and Ruml 2015), since it also introduces other features to the algorithm (i.e. identity action). We show that using single-step error solely, cannot make LRTA* more robust against heuristic inaccuracies caused by read/write error or abstraction. Finally, we evaluated our algorithms not only in terms of the solution suboptimality but also in terms of the GAT-inspired computational effort (GAIO). All pareto-optimal algorithms used the maximum learning weight tried with the number of heuristic retrievals trading solution quality for total planning time.

Acknowledgments

We appreciate help from Shelby Carleton and Delia Cormier and funding from NSERC.

References

Ackley, D. H., and Small, T. R. 2014. Indefinitely scalable computing = artificial life engineering. In *Proceedings of The Fourteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFE 14) 2014*, 606–613. MIT Press.

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

Bulitko, V., and Lee, G. 2006. Learning in real time search: A unifying framework. *Journal of Artificial Intelligence Research(JAIR)* 25:119–157.

Bulitko, V., and Sampley, A. 2016. Weighted lateral learning in real-time heuristic search. In *SoCS*, 10–18.

Bulitko, V. 2016. Evolving real-time heuristic search algorithms. In *Proceedings of the Fifteenth International Conference on the Synthesis and Simulation of Living Systems (ALIFEXV)*, 108–115.

Burns, E.; Kiesel, S.; and Ruml, W. 2013. Experimental real-time heuristic search results in a video game. In *Sixth Annual Symposium on Combinatorial Search (SOCS)*, 47–54.

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. *Proceedings of the IEEE* 98(2):253–266.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions 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. *Journal of Artificial Intelligent Research (JAIR)* 43:523–570.

Hernández, C.; Baier, J.; Uras, T.; and Koenig, S. 2012. Time-bounded adaptive A. In *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, volume 2, 997–1006. International Foundation for Autonomous Agents and Multiagent Systems.

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

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

Korf, R., and Schultze, P. 2005. Large-scale parallel breadth-first search. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 1380–1385.

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

Lelis, L. H.; Valenzano, R.; Nazar, G.; and Stern, R. 2016. Searching with a corrupted heuristic. In *Ninth Annual Symposium on Combinatorial Search (SOCS)*.

Merkle, T.; Knaden, M.; and Wehner, R. 2006. Uncertainty about nest position influences systematic search strategies in desert ants. *Journal of Experimental Biology* 209(18):3545–3549.

O'Ceallaigh, D., and Ruml, W. 2015. Metareasoning in real-time heuristic search. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search (SOCS)*, 87–95.

Piltaver, R.; Luštrek, M.; and Gams, M. 2012. The pathology of heuristic search in the 8-puzzle. *Journal of Experimental & Theoretical Artificial Intelligence* 24(1):65–94.

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

Sadikov, A., and Bratko, I. 2006. Pessimistic heuristics beat optimistic ones in real-time search. *Frontiers in Artificial Intelligence and Applications* 141:148–152.

Schultheiss, P.; Cheng, K.; and Reynolds, A. M. 2015. Searching behavior in social hymenoptera. *Learning and Motivation* 50:59–67.

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

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

Svennebring, J., and Koenig, S. 2004. Building terrain-covering ant robots: A feasibility study. *Autonomous Robots* 16(3):313–332.

Thayer, J. T., and Ruml, W. 2009. Using distance estimates in heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 382–385.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, volume 2, 3.

Traverse, S., and Suave-Hoover, F. 2014. Learning in real time search on robot with limited sensors. Poster, University of Alberta, Department of Computing Science.

Van Oudenhove, L.; Boulay, R.; Lenoir, A.; Bernstein, C.; and Cerda, X. 2012. Substrate temperature constrains recruitment and trail following behavior in ants. *Journal of chemical ecology* 38(6):802–809.