The AIIDE-17 Workshop on
Experimental AI in Games
WS-17-19

# Effects of Self-Knowledge: Once Bitten Twice Shy

**Vadim Bulitko**
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, CANADA
bulitko@ualberta.ca

**Abstract**

Procedurally generating rich, naturally behaving AI-controlled video game characters is an important open problem. In this paper we focus on a particular aspect of non-playable character (NPC) behavior, long favored by science-fiction writers. Specifically, we study the effects of self-knowledge on NPC behavior. To do so we adopt the well-known framework of agent-centered real-time heuristic search applied to the standard pathfinding task on video-game maps. Such search agents normally use a heuristic function to guide them around a map to the goal state. Heuristic functions are inaccurate underestimates of the remaining distance to goal. What if the agent somehow knew how long it (the agent) would actually take to reach the goal from each state? How would using such self-knowledge in place of a heuristic function affect the agent's behavior? We show that similarly to real life, knowing of one's irrational behavior in a situation can deter the agent from getting into that situation again even if it is, in fact, a part of an optimal solution. We demonstrate the "fear" with a simple example and empirically show that the issue is common in video-game pathfinding. We then analyze the issue theoretically and suggest that "fear" induced by self-knowledge is not a bug but a feature and may potentially be used to develop more naturally behaving NPCs.

## 1 Introduction

Non-playable characters (NPCs) are crucial to video games. They fight against the player, cooperate with the player and make the world feel alive. Common techniques for Artificial Intelligence (AI) in NPCs include scripting (McNaughton et al. 2004), finite state machines, behavior trees (Champandard 2007) and, to a lesser degree, automated planning based on heuristic search (Orkin 2006).

Regardless of the AI behind an NPC, players have been conditioned to expect NPCs to act rationally. A hostile NPC is expected to attack the player. A friendly NPC can be expected to co-operate with the player and join his/her quests. Consequently, substantial effort is put into making the NPCs more effective and competent in whatever tasks they are supposed to perform within the game. In the words of Lem (1983), we are building electronic slaves which have no desires, feelings or fears of their own but robotically follow their in-game roles. Real-life characters, whether human or animal, often act irrationally, out of fear or desire. Consider for instance, a character who entered a castle, got lost in it and thus took a long time to get out. It would be natural for a human to regret getting lost and therefore hesitate to enter the castle in the future despite the fact that going through the castle may be the shortest way. Instead, the human is likely chose a path around the castle (Marcatto, Cosulich, and Ferrante 2015). To the best of our knowledge, no existing commercial video-game AI would exhibit such regret-induced non-adaptive choice switching, unless explicitly pre-scripted for a specific context.

In this paper we propose a simple computational approach to having such "once bitten twice shy" behaviors emerge naturally, without any pre-scripting. To make our presentation concrete, we adopt the real-time heuristic search (RTHS) model but, unlike its existing applications, we give our NPCs knowledge about their own behavior. NPCs equipped with such self-knowledge tend to avoid certain areas of the map even if an optimal path to the goal passes through such. As a result, they act irrationally from the pathfinding perspective but arguably more human-like. We conjecture that such behavior will appear more natural and interesting to the player.

The rest of the paper is organized as follows. Section 2 formally introduces the heuristic search problem, presents a common framework of RTHS algorithms, formalizes the notion of self-knowledge and states our performance measures. We then discuss related work in Section 3. Section 4 demonstrates a simple search graph on which self-knowledge leads an agent to acting irrationally. We then show that such irrationality is actually common in the standard grid pathfinding on video-game maps. We then theoretically prove that self-knowledge does help RTHS algorithms that use stationary policies (i.e., hill-climbing) and, in fact, repeated iterations of self-knowledge can lead to convergence to optimal solution costs. Section 5 speculates on the ways the seemingly irrational behavior can make an NPC appear more human-like. We conclude the paper with a discussion of current shortcomings and directions for future work.

## 2 Problem Formulation

Heuristic search is a core subfield of AI and is frequently used for path-finding in video games as well as other types of planning (Orkin 2006). Real-time heuristic search algorithms are often used in the agent-centered environment (Koenig 2001) where the agent has a local view of the world and needs to make decisions before computing a complete solution. In searching for a path on the graph from a start state to a goal state, they occupy a single (current) state and change it by traversing edges. To decide which edge to traverse next, RTHS algorithms conduct a constant-bounded amount of planning, accessing the graph and any information within a constant-bounded neighborhood of the agent's current state. In making the decision they use a heuristic function – an estimate of the cost to-go from a given state to the goal state. The initial heuristic supplied to the agent is usually substantially inaccurate leading to suboptimal decisions made by the agent. Since the seminal work on LRTA* by Korf (1990), many researchers have designed and evaluated approaches to dealing with errors in the heuristic.

Note that while an RTHS agent such as LRTA* does learn as it traverses the search space, it does not exhibit the type of irrational "fear" we discussed in the previous section. That is because LRTA* and similar algorithms update the heuristic in a state based *only* on the heuristic values of the state's neighbors, *ignoring* how they got to that state in the first place. In the castle example, an LRTA* agent would raise the estimated distance-to-exit of a castle location in the same way whether it got to that location after getting lost or walking straight to it.[1]

But what if the agent is aware of its own behavior in an arbitrary state? How would such self-knowledge affect its behavior? In this paper we introduce the idea of self-knowledge into RTHS algorithms and investigate the effects such self-knowledge can have on the agent's behavior. We represent self-knowledge as knowing the precise solution cost of running the agent (self) starting in an arbitrary state while being guided by a heuristic. We then use such self-knowledge in place of the heuristic and observe the resulting behavior. We show that using self-knowledge in this manner can lead to worse paths being found but arguably more human-like behavior.

## 2.1 Search Graph and Performance Measures

We adopt the problem formulation of Bulitko (2016b) and modify it as follows. A *search problem* is a tuple $(S, E, c, s_{\text{start}}, s_{\text{goal}}, h)$ 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*. The search graph is stationary, undirected and *safely explorable* (i.e., the goal state can be reached from any state reachable from the start state). Each edge $(s_a, s_b) \in E$ is weighted by the *cost* $c(s_a, s_b) = c(s_b, s_a) > 0$. The agent begins in the start state $s_{\text{start}}$ and changes its current state by traversing edges (i.e., taking actions) until it arrives at the goal state. In deciding on its actions, the agent has access to a heuristic $h$ which is an estimate of the remaining cost to the goal. We do not assume the heuristic to be admissible or consistent. The agent is free to update it in any way as long as $h(s_{\text{goal}}) = 0$.

The cumulative cost of all edges traversed by a search algorithm $A$ guided by a heuristic $h$ is the *solution cost*, denoted by $C^{A[h]}(s_{\text{start}})$. Note that the search algorithm $A$ starts with the heuristic $h$ but can modify it as it traverses the search graph. The *suboptimality* of a solution, $\alpha(s_{\text{start}})$, is the ratio of the solution cost produced by an agent to the cost of the shortest possible path, $h^*(s_{\text{start}})$. Formally: $\alpha^{A[h]}(s_{\text{start}}) = C^{A[h]}(s_{\text{start}}) / h^*(s_{\text{start}})$. Lower values are desired; the value of 1 indicates solution optimality. If the algorithm $A$ fails to reach $s_{\text{goal}}$ from $s_{\text{start}}$ (even though it is possible due to the safely explorable graph), we say that $C^{A[h]}(s_{\text{start}}) = \infty$. RTHS literature usually focuses on *complete* algorithms which guarantee arriving at the goal state in a safely explorable graph. In this paper we consider incomplete algorithms as well. Given a goal state $s_{\text{goal}}$, define $S^+$ as the set of all states in $S$ from which the goal state is theoretically reachable. Then consider the solution cost for all states in $S^+$. The average suboptimality of such solution costs is $\beta^{A[h]} = \text{avg}_{s \in S^+} \alpha^{A[h]}(s)$.

## 2.2 Real-time Heuristic Search Algorithm

While numerous RTHS algorithms have been proposed, we will demonstrate our self-knowledge experiments primarily on the seminal algorithm, LRTA* (Korf 1990), that started the field and serves as a base for many modern RTHS algorithms (Algorithm 1). As long as the goal state is not reached (line 4) the agent interleaves updating the heuristic with the simple mini-min rule (line 5) and moving to the most promising immediate neighbor (line 6). Ties in $\arg\min$ are broken in an arbitrary order, fixed per state.

**Algorithm 1:** Basic Real-time Heuristic Search

```
input  : search problem $(S, E, c, s_{\text{start}}, s_{\text{goal}}, h)$
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 while $s_t \neq s_{goal}$ do
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     $s_{t+1} \leftarrow \arg \min_{s \in N(s_t)} [c(s_t, s) + h_t(s)]$
7     $t \leftarrow t + 1$
```

## 2.3 Using Self-Knowledge as the Heuristic

Given a state $s$, the heuristic $h(s)$ is an estimate of the remaining solution cost. It is provided as a part of the search problem and guides the agent as it selects its next state (line 6 in Algorithm 1). Instead of using a domain-provided heuristic $h$ for guidance, an algorithm can use the knowledge of its solution cost (with a given heuristic). Such *self-knowledge* can guide the agent in selecting its next state in the same fashion as the usual heuristic. Note that the self-knowledge is the solution cost for the same algorithm but guided by a different heuristic. So the "self-" in the self-knowledge refers only to the algorithm and not the heuristic.

[1]It is possible, however, that all the previous wandering would affect heuristic values of the state's neighbors and subsequently the heuristic value of the state itself upon its update.

Formally, given the solution cost $C^{A[h]}$ for all states from which the goal state is reachable, we can use it as a heuristic within the RTHS algorithm $A$. The resulting solution cost $C^{A[C^{A[h]}]}$ is self-knowledge of the second order and can again be used to guide the agent in place of the regular heuristic. To simplify the notation we define the following iterative process (Algorithm 2).

**Algorithm 2:** Orders of Self-Knowledge

```
input : search problem $(S, E, c, s_{\text{start}}, s_{\text{goal}}, h)$, RTHS algorithm $A$
output: suboptimalities $(\beta_1, \beta_2, \dots)$
1 $C_0 \leftarrow h$
2 $S^+ \leftarrow \{s \in S \mid s_{\text{goal}}$ is reachable from $s\}$
3 $i \leftarrow 0$
4 while not converged do
5     $i \leftarrow i + 1$
6     for $s \in S^+$ do
7         run $A$ from $s$ to $s_{\text{goal}}$ using $C_{i-1}$ as the heuristic
8         $C_i(s) \leftarrow C^{A[C_{i-1}]}(s)$
9     $\beta_i \leftarrow \operatorname{avg}_{s \in S^+} \frac{C_i(s)}{h^*(s)}$
```

On the iteration $i = 1$ the algorithm $A$ is run with $C_{i-1} = C_0 = h$ as its heuristic (line 7 in Algorithm 2). The resulting solution cost for each state in $S^+$ is $C_1 = C^{A[h]}$ (line 8). The average suboptimality of all such costs is $\beta_1$ (line 9). On iteration $i = 2$, we use the self-knowledge $C_1$ as the heuristic and compute $C_2 = C^{A[C_1]} = C^{A[C^{A[h]}]}$. Its average suboptimality is $\beta_2$. The process repeats until $C_i$ converge to a fixed point $C^{\circlearrowleft}$ defined as the cost function that induces itself when used with the algorithm: $C^{A[C^{\circlearrowleft}]} = C^{\circlearrowleft}$. For instance, the perfect heuristic $h^*$ constitutes a fixed point for LRTA*: $C^{\text{LRTA*}[h^*]} = h^*$. We will prove convergence for a specific simple RTHS algorithm (hill-climbing) and evaluate it empirically for other algorithms. In our experiments we detect convergence by computing the change $|\beta_i - \beta_{i+1}|$. If the change remains below $\epsilon$ for $I$ iterations, we declare that the process converged (line 4).

Another question we consider in this paper is whether the average suboptimalities $\beta_i$ monotonically decrease with $i$. Informally, does using (self-)knowledge of $A$'s performance (i.e., solution cost) help $A$ improve its performance? The answer may depend on the search problem, the initial heuristic $h$ and the algorithm $A$ and we will evaluate it empirically in the domain of video-game pathfinding.

Informally, the latter question asks whether knowing about one's behavior may cause one to act less optimally (due to the irrational "fear" of some parts of the search space). The former question asks whether such "fear" can be overcome via higher-order self-knowledge (i.e., knowing about one's behavior with the knowledge of one's behavior).

## 3 Related Work

While Korf (1990) proved that given enough learning trials LRTA* will converge to an optimal solution, the convergence process can be non-monotonic. Specifically, an LRTA*-controlled agent will often follow a short solution found on a trial with a much longer one on the next trial (Shimbo and Ishida 2003, Figure 2). This is due to the fact that the initial heuristic is typically admissible and thus underestimates the true distances to the goal. The learning rule (line 5 in Algorithm 1) raises the heuristic in the states visited by the agent which consequently makes the states not yet visited look more appealing (line 6) and thus guides the agent away from a good path already found.

The phenomenon studied in this paper is only superficially similar. Indeed, in Korf's multi-trial learning process the heuristic LRTA* receives on a trial is the heuristic LRTA* used and updated in some states on the previous trial. The non-monotonicity of solution suboptimality over subsequent learning trials is partly due to updating the heuristic in some but not all states. In contrast, self-knowledge used by LRTA* on an iteration (trial) is computed on the previous iteration for *all* states (line 6 in Algorithm 2). Furthermore, the heuristic of each state is computed by running LRTA* from it to the goal, independently of LRTA* runs originating in other states. In other words, for any two states $s_1, s_2 \in S^+$, $C_i(s_1)$ is computed independently from $C_i(s_2)$. Furthermore, any heuristic updates LRTA* made when launched from state $s_1$ are discarded before it is launched again from the state $s_2$. In fact, our implementation computes $C_i(s)$ in a parallel loop over $s$.

Using solution costs $C_i$ to guide an agent is superficially connected to influence maps (Tozour 2001). There are several key differences. We do not seed the influence map at certain cells and then propagate/blur it to neighboring cells. We compute each cell's value completely independently of the others. We also do not use $C_i$ of a state as its desirability for the agent. Instead $C_i$ are used as heuristic values which are then combined with edge costs and used to select decisions in an arbitrarily complex fashion, as determined by a given heuristic search algorithm. Finally, by treating $C_i$ as a heuristic, the agent itself can update it during a single trial.

The first iteration of Algorithm 2 (i.e., using $C_1$ in place of the heuristic $h$) can be viewed as a special case of rollout-based approaches such as UCT (Kocsis and Szepesvári 2006). Such approaches usually use a randomized rollout policy to estimate the value of a state from multiple rollouts launched from that state. In our case, the solution cost $C_1 = C^{A[h]}$ gives the value of any state from a single run of the deterministic rollout policy $A[h]$. There are two differences in our approach: we compute $C_i$ for $i > 1$ and we use a single run of an arbitrarily complex rollout policy $A[C_{i-1}]$.

Another area of related work is lookahead in heuristic search. Korf (1990) built a fixed-depth lookahead tree from the agent's current state and backed up heuristic values of the leaf states. Subsequent research experimented with different lookahead tree shapes and backup rules (Koenig and Sun 2009; Koenig and Likhachev 2006) as well as state-adaptive lookahead (Bulitko et al. 2008; O'Ceallaigh and Ruml 2015). Our approach is different. First, our backed up value of a state is determined from a single complete path from it to the goal. Second, we build a sequence of $C_i$ for $i \geq 1$. Third, $C^{A[h]}$ is computed for each state independently using the "clean" heuristic $h$, whereas backed up value from a lookahead tree can depend on the previous updates to the heuristic. Interestingly, lookahead pathologies where deeper lookahead leads to worse moves have been observed (Luštrek and Bulitko 2006).

Finally, most existing RTHS research has focused on finding high-performance algorithms, either crafted by hand (Ishida 1992; Shue and Zamani 1993a; 1993b; Shue, Li, and Zamani 2001; Bulitko 2004; Hernández and Meseguer 2005; Bulitko and Lee 2006; Rayner et al. 2007; Bulitko et al. 2007; Koenig and Sun 2009; Sturtevant, Bulitko, and Björnsson 2010; Sturtevant and Bulitko 2011; Hernández and Baier 2012; Sharon, Sturtevant, and Felner 2013; Rivera, Baier, and Hernández 2015; O'Ceallaigh and Ruml 2015) or found by an automated search through the space of algorithms (Bulitko 2016a; 2016c). Our work is complimentary as it evaluates impact of using self-knowledge and can be done with any RTHS algorithm.

## 4 Policy Improvement/Degradation

In Section 2.3 we formulated our first question as whether the average suboptimalities $\beta_i$ monotonically decrease with $i$. In other words, does using the self-knowledge $C_i$ as a heuristic reduce the solution cost $C_{i+1}$ (on average)?

One may think that the answer is Yes due to Sutton and Barto (1998). Ported from Markov decision processes (MDP) used in reinforcement learning (RL) to the deterministic shortest path problem on graphs, the result becomes:

**Theorem 1 (Policy improvement).** Given a complete stationary agent-control policy $\pi : S \to S$ which deterministically maps any state $s$ to one of its immediate neighbors $\pi(s) \in N(s)$, the control policy $\pi'$ greedy with respect to the solution cost of $\pi$: $\forall s \in S^+ \left[\pi'(s) = \arg\min_{s' \in N(s)} [c(s,s') + C^\pi(s')]\right]$ is at least as good as $\pi$: $\forall s \in S^+ \left[C^{\pi'}(s) \le C^\pi(s)\right]$.

**Proof.** Adapting the proof by Sutton and Barto (1998), for any state $s \in S^+$ we have $C^\pi(s) = c(s,\pi(s)) + C^\pi(\pi(s)) \ge \min_{s' \in N(s)} [c(s,s') + C^\pi(s')] \ge c(s,\pi'(s)) + C^\pi(\pi'(s)) \ge c(s,\pi'(s)) + c(\pi'(s),\pi'(\pi'(s))) + C^\pi(\pi'(\pi'(s))) \ge \dots \ge C^{\pi'}(s)$.■

Despite the theorem, there is no guarantee of monotonic solution-cost improvement with even basic RTHS algorithms, as shown below.

### 4.1 Counter Example

The example in Figure 1 shows that self-knowledge can actually hurt LRTA* solution cost. This is an eight-connected grid with the goal shown in green (and marked with 0 in it). Blocked cells are shown in dark grey. The intensity of red and the labels in all non-goal cells show $C_i$. The left most plate lists the initial heuristic $h$ values (octile distance). With the initial heuristic LRTA* makes a mistake in the state with a dotted blue frame, resulting in a high solution cost of 10.4 (second plate). This means that LRTA* run with $C^{\text{LRTA*}[h]}$ as its heuristic will avoid that state because it thinks it would behave badly in it.[2] This is the "bitten" part of "once bitten twice shy". Note that this "fear" of the state is irrational insomuch as the state lies on the only optimal path to the goal for all states below it and thus must be visited.

As the third plate shows, the irrational avoidance of the framed state leads to higher solution costs for the states below it than they were when LRTA* used the vanilla $h$. For instance, 5.41 becomes 10.8. Even averaged over all states, the solution cost of LRTA* using self-knowledge is higher than the solution cost of LRTA* using $h$: average suboptimality $\beta_2 = \beta^{\text{LRTA*}[C^{\text{LRTA*}[h]}]} = 1.394$ while average suboptimality $\beta_1 = \beta^{\text{LRTA*}[h]} = 1.253$.[3]

There are two ways to think about this degradation of LRTA* performance due to self-knowledge. First, the knowledge that LRTA* does poorly when launched from a state can mean two things: (i) the state is bad and should indeed be avoided or (ii) the state is good but an inaccurate heuristic resulted in an inflated solution cost from that state. LRTA* using self-knowledge as its heuristic is unable to distinguish between these two cases and thus can be "shy" of any state where it was "bitten".

The second way of thinking about the performance degradation lies with the notion of heuristic depressions (Ishida 1997) which has been subject of recent research (Hernández and Baier 2014; Sturtevant and Bulitko 2016; Bulitko and Sampley 2016). Viewed in terms of heuristic depressions, the degradation of LRTA* with self-knowledge can be viewed as follows. The initial heuristic had a depression with the local minimum of $h = 2$ (left-most plate). The solution cost $C_1 = C^{\text{LRTA*}[h]}$ also has a depression with the local minimum in the same state but the rim of the depression is now higher: 10.4 instead of 2.41. Thus, the LRTA* with self-knowledge spends more time filling its heuristic depression before it can escape it. Consequently, LRTA* performs worse with the self-knowledge $C_1$ than it did with the initial octile distance heuristic $C_0$.

### 4.2 Generality of the Problem

Perhaps the example in the previous section is pathological and self-knowledge normally does help RTHS algorithms? We will now show that this does not seem to be the case.

**LRTA*.** We ran the basic LRTA* (Algorithm 1) on 8 maps from the MovingAI repository (Sturtevant 2012). Figure 2 shows the average suboptimality of the solution cost $\beta_i$ as a function of the iteration $i$, run until convergence.The averages and standard deviations are computed over 512 goals.[4]

[2]Note that a high $C^{\text{LRTA*}[h]}(s)$ means merely that LRTA* will behave poorly when launched from $s$. It does not necessarily mean that LRTA* will behave poorly in $s$ when it comes to it launched from another state. Indeed, by the time LRTA* reaches $s$ after being launched from another state, it may have updated its heuristic to preclude poor behavior in $s$.

[3]Note that $C_2 = C^{\text{LRTA*}[C^{\text{LRTA*}[h]}]}$, while high, has only a single minimum at the goal state. This means that LRTA* ran with $C_2$ as its heuristic is optimal and indeed $C_3 = C^{\text{LRTA*}[C_2]} = h^*$ (right-most plate in the figure). Consequently, $\beta_3 = 1$.

[4]The maps were `012`, `014`, `307`, `603`, `701` from the game *Baldurs Gate II* and `darkforest`, `divideandconquer`,

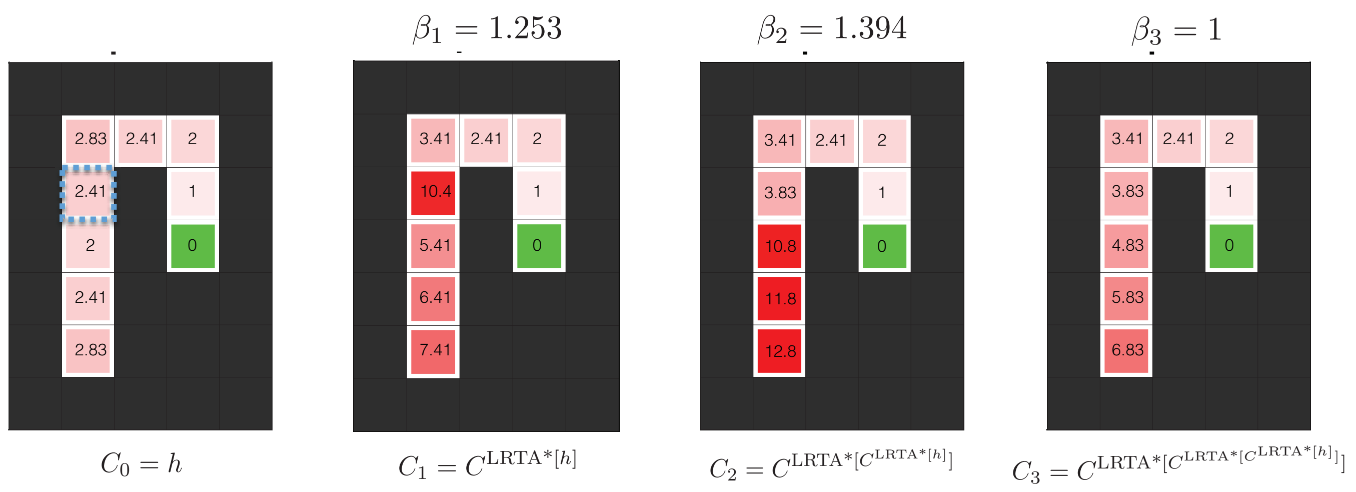

Figure 1: Self-knowledge hurts LRTA*.

The peak in the averages suggests that self-knowledge hurts suboptimality on early iterations.

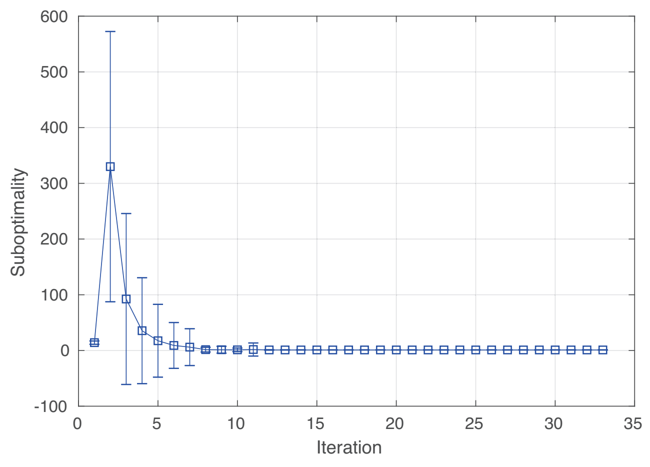

Figure 2: Effects of self-knowledge with LRTA*.

**Generalized LRTA*.** But perhaps the simple LRTA* is somehow a pathological algorithm and its more modern descendants are not subject to the harmful effects of self-knowledge? To answer this question we repeated the experiment above but averaged $\beta_i$ over algorithms randomly sampled from a generalization of the recently introduced space of RTHS algorithms (Bulitko 2016c). We added a step size parameter to the learning rule, randomly generated such algorithm specifications and ran each of them in place of LRTA* in a similar setup. As with LRTA*, the values of $\beta_i$ averaged over the random algorithms rise on early iterations, indicating that the phenomenon is not restricted to LRTA*.

`thecrucible` from the game *Warcraft III*. We capped suboptimality of any run at $10^3$. We deemed the process to converge when $\beta_i$ did not change more than $\epsilon = 10^{-3}$ over $I = 10$ consecutive iterations. The 64 goal states on each of the 8 maps were chosen randomly but so that $\beta_1 \in [10, 20]$ for each goal.

## 4.3 Policy Improvement with Hill Climbing

The reason that policy iteration may not necessarily improve policy value $C^\pi$, seemingly in contradiction to Theorem 1, is that LRTA* and other LRTA*-like algorithms are not a (stationary) policy $\pi$ as defined in the theorem. While LRTA*-like algorithms do act greedily with respect to its heuristic function and the edge costs (line 6 in Algorithm 1), they also change the heuristic over time (line 5). Thus, an LRTA*-controlled agent may not follow any one policy $\pi$ as defined in the theorem.

Conversely, LRTA* without learning (i.e., with line 5 disabled) does implement a stationary policy – greedy with respect to its now stationary heuristic – and is therefore a subject to Theorem 1:

**Corollary 1 (Policy improvement with hill climbing).** Define a hill-climbing control policy $\text{HC}[h]$ greedy with respect to a heuristic $h$ as: [5]

$$\forall s \in S \left[ \text{HC}[h](s) = \arg \min_{s' \in N(s)} [c(s, s') + h(s')] \right]. \qquad (1)$$

Suppose the search graph is such that an agent following $\text{HC}[h]$ reaches the goal state starting from any state $s$: $\forall s \in S^+ \left[ C^{\text{HC}[h]}(s) < \infty \right]$. Then the hill-climbing policy HC can be improved via policy iteration: $\forall s \in S^+ \left[ C^{\text{HC}[C^{\text{HC}[h]}]}(s) \leq C^{\text{HC}[h]}(s) \right]$.

**Proof.** Define $\pi(s) = \arg\min_{s' \in N(s)} [c(s, s') + h(s')]$. Then the desired inequality follows directly from Theorem 1. ■

Informally, this result means that self-knowledge allows a hill-climbing agent to improve. Note that the proof of Theorem 1 required finite solution costs which means that the

[5]Such a policy is called *hill climbing* as the agent (down-) climbs the surface of $h$ until it reaches a local optimum.

agent must be able to hill-climb to a goal state from any state in $S^+$. In other words, the heuristic surface should be fully devoid of local optima which is unrealistic for any non-trivial search graph and a reasonable heuristic.

## 4.4 Convergence to Optimal Cost

We will now consider the more realistic case where a hill-climbing agent cannot reach the goal state from some states in $S^+$ given its initial heuristic: $\exists s \in S^+ \left[C^{\mathrm{HC}[h]}(s) = \infty\right]$. Functionally this means that hill-climbing from such a state $s$ the agent reaches a local minimum in the heuristic surface and stays there forever without ever reaching the goal (since HC does not modify heuristic). Can self-knowledge help? To answer this question we first generalize hill-climbing to infinite heuristic values. Specifically, if all neighbors of the current state have infinite values of $h$ then the $\arg\min$ operator in Equation (1) considers them tied and uses a deterministic tie-breaking schema. We can now prove:

**Theorem 2** For any initial heuristic satisfying the conditions of Section 2, after a finite number of iterations of Algorithm 2 with the hill-climbing agent, the iteration cost $C$ converges to the optimal cost $h^*$ for all connected states: $\exists J \forall j \geq J \forall s \in S^+ \left[C_j(s) = h^*(s)\right]$.

**Proof.** First, we define a set of states: $F_i = \{s \in S^+ \mid C_i(s) = C^{\mathrm{HC}[C_{i-1}]}(s) = h^*(s)$ and HC launched in the state $s$ with heuristic $C_i$ follows an (optimal) path entirely contained in $F_i\}$, $i \geq 1$. Note that $s_{\text{goal}}$ and all its immediate neighbors are members of $F_1$ since HC$[h]$ reaches the goal optimally from any of them. We will now prove that once a state enters $F_i$, it will be in $F_j$ for any $j \geq i$. Suppose this is not the case. Then there exists $F_i$ and a state $s \in F_i$ such that $s \notin F_{i+1}$. As $s \in F_i$ the hill-climbing agent guided by $C_{i-1}$ travels from $s$ to $s_{\text{goal}}$ along an optimal path $P$ fully contained in $F_i$. As $s \notin F_{i+1}$, the hill-climbing agent guided by $C_i$ deviates from the path $P$. Let state $s'$ be the earliest state in which the deviation occurs (Figure 3, left). While in $s'$, HC$[C_{i-1}]$ chose $s'' \in P$. On the other hand HC$[C_i]$ located in $s'$ chose $s''' \notin P$. Given how HC chooses its states (Equation 1) and the fixed tie-breaking we use, the following inequality must hold for the switch from $s''$ to $s'''$ to happen: $c(s', s''') + C_i(s''') < c(s', s'') + C_i(s'')$. Since $s'' \in F_i$ it follows that $C_i(s'') = h^*(s'')$: $c(s', s''') + C_i(s''') < c(s', s'') + h^*(s'')$ which is impossible since $C_i(s''') \geq h^*(s''')$ and $s''$ is on an optimal path from $s'$ to the goal. This means that HC$[C_i]$ will not change its behavior when launched in $s$ and will stay on $P$. Thus, $s \in F_{i+1}$ which contradicts our supposition. Hence, no state can leave $F_i$ and $F_i \subset F_{i+1}$.

We will now show that $F_i$ necessarily grows by at least one state with each iteration until it absorbs all states in $S^+$. Suppose $F_i \neq S^+$ then there must be a state $s \in S^+ \setminus F_i$ such that an optimal path from $s$ to $s_{\text{goal}}$ enters $F_i$ with its very first edge (Figure 3, right). If all of $s$'s neighbors that lie on optimal paths to the goal are already in $F_i$ then $s$ will necessarily enter $F_{i+1}$. Suppose, however, that $s$ has a neighbor $s''$ which is on an optimal path to goal but is outside of $F_i$. If HC$[C_i]$ chooses to go from $s$ to $s'$ then

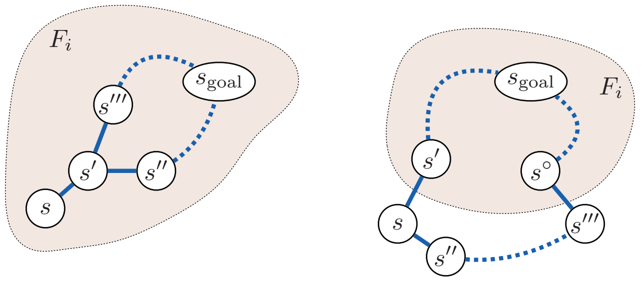

Figure 3: To the proof of Theorem 2.

$s \in F_{i+1}$. Suppose HC$[C_i]$ chooses to go from $s$ to $s''$. Then:

$$c(s, s'') + C_i(s'') \leq c(s, s') + C_i(s') \quad (2)$$
$$c(s, s'') + C_i(s'') \leq c(s, s') + h^*(s') \quad (3)$$
$$c(s, s'') + h^*(s'') = c(s, s') + h^*(s') \quad (4)$$

which means that $C_i(s'') = h^*(s'')$ and $s''$ is also on an optimal path from $s$ to goal. Tracing this optimal path $P = (s, s'', \dots, s''', s^\circ, \dots, s_{\text{goal}})$ we see that it enters $F_i$ with the edge $(s''', s^\circ)$ where $s''' \notin F_i$ but $s^\circ \in F_i$. This means that HC$[C_i]$ launched in the state $s'''$ follows the optimal path $(s''', s^\circ, \dots, s_{\text{goal}})$ which means that $s'''$ enters $F_{i+1}$. This means that $F_i$ never loses states and grows by at least one state until it becomes $S^+$ at iteration $J$. ■

We just showed that despite the fact that a hill-climbing agent is initially incomplete and never changes its heuristic, in at most $|S^+|$ iterations of Algorithm 2 it will be able to reach the goal from any connected state and do so optimally. While empirical results suggest that the same holds for self-knowledge iterations with heuristic-updating agents such as LRTA* proving so is a subject of future work.

## 5 Human-like Irrational Behavior

Using solution cost as a heuristic can discourage the agent from visiting any states where it knows to perform poorly (with the original heuristic). Just as with the proverb "Once bitten twice shy", the agent will avoid such "bite" states even if they are indeed on an optimal path to the goal. On the other hand, knowing that a solution cost in a state is high can be useful when the state is not on an optimal path to goal and thus should indeed be avoided. Such solution costs are especially helpful when the initial heuristic is low and thus invites the agent to visit the useless state.

How can an agent tell between the two scenarios? Does the ranking of a state's neighbors induced by a solution cost with a heuristic indicate the true ranking of such neighbors? In using solution cost from a previous iteration as the guiding heuristic for the next iteration, our agents can assume so even when it is not the case.

Such irrational behavior is related to the distinction between adaptive/rational and non-adaptive/irrational choice switching. Marcatto, Cosulich, and Ferrante (2015) showed that regret from being under the (false) impression that an objectively suboptimal choice would have been better can prevent humans from making the optimal choice on the subsequent trials. While the experimenters set it up by misinforming the participants of the outcome of the choice they did not make, our agents are effectively misinformed by basing their choice on the solution length with an inaccurate heuristic. In both cases, human/AI agents use past outcomes to shape their future choices even when the future and the past experiences are not aligned. Thus our NPCs can display the specific type of human irrational behavior (non-adaptive choice switching) without being explicitly scripted to do so.

## 6 Current Limitations and Future Work

The study presented in this paper is the first look at the use of self-knowledge in RTHS. As such it opens many exciting directions for future work. First, our conjecture that the irrational fear due to self-knowledge makes an NPC more natural in the eyes of the player needs to be confirmed/refuted via a user study. Second, where does such self-knowledge come from? In our experiments we computed it by independently running LRTA* from every state with a fresh heuristic and recording its solution cost. While this is prohibitively expensive for large maps, there can be more practical ways of obtaining such self-knowledge. For instance, an agent can observe a solution produced by another agent (social learning) or the agent can recall its own solution quality for similar states and combine them into an estimate of its solution quality from a novel state (case-based reasoning).

## 7 Conclusions

To the best of our knowledge, this paper is the first investigation into the use of self-knowledge in real-time heuristic search. We showed that using solution costs in place of the usual heuristic can harm solution quality. We demonstrate this curious fact on a simple grid-pathfinding example and empirically showed that this is a common phenomenon with LRTA* as well as more general RTHS algorithms sampled from a large space of algorithms. We proved that for hill-climbing the solution costs do converge to optimal. Finally, we conjectured that embracing the "once bitten twice shy" paradigm can make NPCs appear more human-like in game.

## 8 Acknowledgements

We appreciate discussions with Valeriy Bulitko, feedback from Shelby Carleton, Delia Cormier and John Simpson and funding by NSERC.

## References

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

Bulitko, V., and Sampley, A. 2016. Weighted lateral learning in real-time heuristic search. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 10–18.

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

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

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

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

Bulitko, V. 2016b. Per-map algorithm selection in real-time heuristic search. In *Proceedings of Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 143–148.

Bulitko, V. 2016c. Searching for real-time search algorithms. In *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 121–122.

Champandard, A. 2007. Behavior trees for next-gen game ai. In *Game Developers Conference, Audio Lecture*.

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

Hernández, C., and Baier, J. A. 2014. Avoiding and escaping depressions in real-time heuristic search. *CoRR* abs/1401.5854.

Hernández, C., and Meseguer, P. 2005. LRTA*(k). In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1238–1243.

Ishida, T. 1992. Moving target search with intelligence. In *Proceedings of the National Conference on Artificial Intelligence*, 525–532.

Ishida, T. 1997. *Real-time Search for Learning Autonomous Agents*. Kluwer Academic Publishers.

Kocsis, L., and Szepesvári, C. 2006. Bandit based monte-carlo planning. In *Proceedings of the European Conference on Machine Learning (ECML)*, 282–293.

Koenig, S., and Likhachev, M. 2006. Real-time adaptive A*. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 281–288.

Koenig, S., and Sun, X. 2009. Comparing real-time and incremental heuristic search for real-time situated agents. *Journal of Autonomous Agents and 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. *Artificial Intelligence* 42(2–3):189–211.

Lem, S. 1983. Doctor Diagoras. In *Memoirs of a space traveler: further reminiscences of Ijon Tichy*. Houghton Mifflin Harcourt.

Luštrek, M., and Bulitko, V. 2006. Lookahead pathology in real-time path-finding. In *Proceedings of the National Conference on Artificial Intelligence (AAAI), Workshop on Learning For Search*, 108–114.

Marcatto, F.; Cosulich, A.; and Ferrante, D. 2015. Once bitten, twice shy: experienced regret and non-adaptive choice switching. *PeerJ* 3:e1035.

McNaughton, M.; Cutumisu, M.; Szafron, D.; Schaeffer, J.; Redford, J.; and Parker, D. 2004. Scriptease: Generative design patterns for computer role-playing games. In *Proceedings of the 19th IEEE International Conference on Automated Software Engineering*, ASE '04, 88–99. Washington, DC, USA: IEEE Computer Society.

O'Ceallaigh, D., and Ruml, W. 2015. Metareasoning in real-time heuristic search. In *Proceedings of the Eighth Annual Symposium on Combinatorial Search, SOCS 2015, 11-13 June 2015, Ein Gedi, the Dead Sea, Israel.*, 87–95.

Orkin, J. 2006. Three states and a plan: the AI of FEAR. In *Game Developers Conference*, 4.

Rayner, D. C.; Davison, K.; Bulitko, V.; Anderson, K.; and Lu, J. 2007. Real-time heuristic search with a priority queue. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2372–2377.

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

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

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

Shue, L.-Y., and Zamani, R. 1993a. An admissible heuristic search algorithm. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, 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 *Proceedings of the International Joint Conference on Artificial Intelligence*, 365–370.

Sturtevant, N., and Bulitko, V. 2016. Scrubbing during learning in real-time heuristic search. *Journal of Artificial Intelligence Research* 307–343.

Sturtevant, N. R.; Bulitko, V.; and Björnsson, Y. 2010. On learning in agent-centered search. In *Proceedings of the Conference on Autonomous Agents and Multiagent Systems*, 333–340.

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

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Tozour, P. 2001. Influence mapping. In *Game Programming Gems*, volume 2. Charles River Media. 287–297.