Evolving Initial Heuristic Functions for Agent-Centered Heuristic Search

Vadim Bulitko
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
bulitko@ualberta.ca

***Abstract*—Heuristic functions guide search algorithms and have a profound impact on their performance. In the context of agent-centered real-time heuristic search (RTHS), a heuristic represents the agent's initial domain knowledge which the agent then updates as it explores the search graph. An ideal initial heuristic should capture some specific domain knowledge to guide the agent effectively yet be general enough for a broad class of problems. It should also be computationally efficient, compact in its representation and human-interpretable. Traditionally initial heuristics in RTHS have been designed by humans (e.g., Manhattan distance). In this paper we explore the alternative of building initial heuristics by machines. To keep them portable and human-interpretable we represent each heuristic as a closed-form algebraic formula. Yet to make the heuristics capture problem specifics and thus be more effective in guiding the search, we automatically build a heuristic tailored to a class of problems. To achieve both objectives, we propose and evaluate automatically searching the space of heuristic functions. As a preliminary demonstration, we find closed-form heuristics that outperform Manhattan distance in grid-based pathfinding. We then develop an insight on how such formula-based heuristics are able to exploit characteristics of certain pathfinding maps.**

***Index Terms*—heuristic search, real-time heuristic search, evolution, heuristic function**

I. Introduction

Heuristic search is a classic and fundamental part of Artificial Intelligence. In finding a shortest path from a start vertex to a goal vertex in a search graph, heuristic search uses a heuristic (function), an estimate of distance remaining to goal, for guidance in exploring the graph. A good heuristic substantially reduces the amount of search needed while providing guarantees on the resulting path cost. Real-time heuristic search (RTHS) makes the search agent-centered [1] which is useful when the agent needs to traverse a search graph on-line, using information local to it and facing time pressure. The classical application is pathfinding in video games. Numerous RTHS algorithms have been proposed and studied since the seminal LRTA* [2]. Most of them use a graph-independent human-designed heuristic (e.g., Manhattan distance in grid-based pathfinding). However, such generic human-designed heuristics often ignore specifics of a search graph (e.g., Manhattan distance ignores map structure in pathfinding which misleads the agent). Thus RTHS algorithms commonly update the initial inaccurate heuristic on-line using a variation of the Bellman optimality equation (also known as the mini-min rule). The updates can be slow and cause behaviour that appears irrational to a user [3]. Thus, most of RTHS research in the past thirty years has focused on developing better RTHS algorithms which more effectively operate with a standard initial heuristic. In this paper we take the complementary angle of keeping the RTHS algorithm fixed but building a better initial heuristic. We do so by automatically searching through a space of initial heuristics.

There are three ways to frame our approach. **First**, from an application perspective, one can imagine automatically fitting a heuristic to a class of problems. For instance, a video game may use maps that share some characteristics (e.g., narrow hallways). Thus, a video-game developer may opt to find a heuristic that captures such map characteristics and thus allows for a better pathfinding with an off-the-shelf algorithm. As long as the heuristic is portable it can be also effective on maps that change during the gameplay as well as maps built by players after the game is released.

Second, from a machine-learning perspective, an RTHS agent performs on-line learning. A heuristic-learning RTHS algorithm such as the classical LRTA* is similar to temporal-difference algorithms in reinforcement learning (RL). Indeed, it updates its estimate of the remaining distance (the heuristic) using its experience (the cost of the edges around it) and its other estimates (heuristic values of neighbouring states). The question is when/where the learning happens. In RL it is common to start with a random value function which means that all learning is done by the agent on-line. In RTHS it is common to start with a human-designed, reasonable heuristic and then let the agent update it on-line to fit the search problem at hand. Our approach provides the third option: spread the learning over generations of agents so that later agents start with a more effective initial (i.e., "innate") heuristic. A related idea – generational learning of effective reward functions and initial policies – has been successfully implemented in RL [4]. The important issue is the balance of heuristic's effectiveness and portability. Manhattan distance is very portable (i.e., applies to any four-connected grid-based pathfinding map) but not very effective as it ignores all map structure. A perfect heuristic completely captures such a structure but only for a single map which makes it misleading on other maps. Our approach automatically learns heuristics that are between the two extremes.

Third, there has recently been a resurgence of interest in Artificial General Intelligence. Within the larger field of heuristic search, RTHS is agent-centered [1] and bounded in its rationality [5] due to the real-time constraint. Thus, RTHS is arguably more closely related to human-decision making than general heuristic search. Indeed, recent RTHS work made connections to human decision-making traits such as self-reflection [6] and anxiety [7]. Continuing with such connections one can argue that the two cognitive systems that humans appear to use for decision-making [8] have natural counterparts in RTHS. Specifically, human cognitive System 1 is fast and intuitive but often makes mistakes. System 2 is slow and analytical and overrides System 1 when the person makes a decision to do so. In RTHS agents, System 1 corresponds to acting greedily with respect to the current heuristic function. It is fast as only immediate neighbours of the agent's current state need to be considered but often makes mistakes since the heuristic is often inaccurate [3]. System 2 is the lookahead which is computationally more expensive but generally makes better decisions [2], [9]. An agent can opt to increase its lookahead — engage System 2 — in certain contexts such as a heuristic depression [10]. The work in this paper furthers the parallel between human Systems 1 and 2 and RTHS by considering how the heuristic function can emerge over generations of RTHS agents.

The paper makes the following contributions: it formulates a novel problem in the field of RTHS, it proposes an application of progressive evolution to automatically fit a heuristic function to a class of problems, it presents a preliminary empirical evaluation on grid-based pathfinding and a preliminary insight into evolved heuristics.

II. Problem Formulation

A. Real-time Heuristic Search

In this paper we adapt the standard definition of RTHS [7]. The agent is traversing an undirected search graph $G = (S, E)$ comprised of a finite set of vertices/states S connected by a finite set of edges $E \subset S \times S$. Each edge is weighted by a cost function $c : E \to \mathbb{R}$. All costs are positive. There are no self-loops in the graph. Time proceeds in discrete steps. At time t the agent occupies a single state s_t which it changes to a neighbouring state $s_{t+1} \in N(s_t)$ by traversing the edge $(s_t, s_{t+1}) \in E$. Neighbours of state s are denoted by $N(s)$. The agent starts in a start state s_0 and eventually arrives in the goal state s_g. The agent's solution is the path $(s_0, s_1, \dots, s_g)$. The cumulative cost of all edges in that path is the *solution cost*. The *solution suboptimality* is then the ratio of the solution cost to the lowest possible solution cost. Solution suboptimality of 1 indicates an optimal solution while solution suboptimality of 2 indicates a path twice as costly as optimal.

While traversing G, the agent is guided by a *heuristic function* $h : S \to \mathbb{R}$ which estimates the remaining cumulative travel cost between a state and the goal. The search is, however, real-time and the agent is allowed to consider only up to k states in the graph before it is required to traverse an edge and k is independent of the graph size $|S|$. Additionally, in agent-centered search the states considered by the agent in deciding on the edge to traverse are required to be around the agent's current state. Since the agent is required to act before it can compute a complete solution, state revisits are likely. To avoid looping forever, the agent updates its heuristic. The initial heuristic h_0 is an input to the agent.

B. Search for Initial Heuristics

The initial heuristic h_0 has a fundamental effect on suboptimality of the solution produced by the agent. For instance, the perfect heuristic h^* that gives the cost of the optimal path between each state and a goal is guaranteed to produce an optimal solution with many RTHS algorithms while the zero initial heuristic $\forall s \in S\, [h_0(s) = 0]$ provides no guidance at all, likely leading to highly suboptimal solutions. Given a problem instance $p = (G, c)$ we denote the solution cost found by the agent a starting with the initial heuristic h_0 by $C(p, a, h_0)$. Its solution suboptimality is then $\alpha(p, a, h_0) = C(p, a, h_0)/C^*(p)$ where $C^*(p)$ is the lowest solution cost possible for p.

The problem addressed in this paper is thus to find an initial heuristic $h_0^{\min}$ that minimizes the solution suboptimality of the algorithm a averaged over a set of problems P: $h_0^{\min} = \operatorname*{argmin}_{h_0 \in \mathcal{H}} \operatorname*{mean}_{p \in P} \alpha(p, a, h_0)$ where $\mathcal{H}$ is a space of initial heuristic functions. Note that with poor heuristics an agent may travel a long time before arriving at the goal state which makes computing $C(p, a, h_0)$ expensive. Thus, in line with related prior work in the field [11], [12] we stop the agent as soon as its travel cost exceeds $\alpha_{\max} C^*(p)$ where $\alpha_{\max} \geq 1$ is a parameter (suboptimality cap). The resulting truncated travel cost is used in computing $\alpha(p, a, h_0)$. Naturally this is only possible when $C^*(p)$ is known for a problem p.

C. Desired Properties of a Solution

The heuristic $h_0^{\min}$ can capture some properties of search problems in the set P and guide an RTHS algorithm better. Below we list several beneficial properties of $h_0^{\min}$ as well as of the process of computing it.

Automatic Derivation. We would like $h_0^{\min}$ to be found automatically for a given class P of search problems and a given RTHS algorithm. One way to do so is to search for $h_0^{\min}$ to fit a random subset P' of the target problem set P. A good approach will then avoid overfitting to P' and generalize onto the larger set P.

Explainability. Towards explainable AI we would like $h_0^{\min}$ to be human-interpretable, similarly to Manhattan distance and in contrast to larger deep neural networks. The benefits of human interpretability include an insight into design of better heuristics and explainable AI.

Compact Representation. We would like $h_0^{\min}$ to have a compact representation. This may help make it computationally inexpensive and human-interpretable. This is also important in an A-life setting where the initial heuristic would be encoded genetically and passed on from an agent to its offspring [4]. Larger representations would then require larger genomes and may be more vulnerable to random mutations.

Low Computational Cost. We would like $h_0^{\min}$ to be easily computable by the agent. Doing so allows the agent to increase the amount of its local planning before it is forced to make a move. Previous work in real-time heuristic search has shown less accurate but faster to compute/learn heuristics to be advantageous [13].

Portability. An initial heuristic function can apply to one or more search problems. For instance, in grid-based pathfinding the commonly used Manhattan and Euclidean heuristics are represented by map-independent formulae and thus can be applied to any grid map. On the other hand, the perfect heuristic h^* is a table of values thus specific not only to a map but also to the goal state. We would like to strike a balance between the extremes and have a single initial heuristic h_0 that gives a search agent useful guidance on many search problems.

III. Related Work

LRTA* guided by an inaccurate heuristic is prone to state re-visitation known as scrubbing [14] which happens when the agent gets temporarily stuck in a heuristic depression. While various extensions to LRTA* can sometimes allow the agent to escape a heuristic depression faster [3], a better heuristic does not create heuristic depressions in the first place. Past work built better heuristics in the following ways.

First, an RTHS agent can improve the initial heuristic faster by replacing the basic mini-min learning rule [2] with more aggressive learning mechanisms [15], [16]. It is possible that such mechanisms help because they implicitly capture properties specific to a given class of search problems [11]. If so this is a dual and complementary approach to ours.

Second, common human-designed heuristics tend to ignore some properties of a search problem (e.g., Manhattan distance ignores any obstacles on a pathfinding map or interactions between tiles in Puzzle 15). Thus, heuristic values of states closer to the goal tend to be more accurate as there is less to ignore. Consequently, one can improve heuristic quality by replacing the original distant goal with a near-by subgoal. Such approaches commonly pre-compute a problem-specific and thus not portable set of subgoals [17]–[22].

Third, instead of routing the agent to near-by subgoal, one can use subgoals for estimating remaining distance with respect to the original goal [23], [24]. These approaches pre-compute information (e.g., distances between gateways) specific to a search graph, forgoing portability. Also heuristics for non-real-time search algorithms such as A* are typically designed to be admissible in order to preserve solution optimality. This is not a consideration in RTHS and, in fact, inadmissible heuristics can be desirable there [25], [26].

Fourth, one can use abstractions of the original search problem to build a better heuristic represented as a pattern database [27], [28]. It can deliver a great improvement in search efficiency but the resulting database is usually specific to a search graph and a particular goal in it. Furthermore, effective pattern databases can be large, contrary to our desire for compactness and human readability.

Finally, capturing properties of a class of search problems can be viewed as a compression of full descriptions of the problems into something more compact and useful to the agent in its search. Prior work [20], [21] compressed a series of pre-computed paths by keeping only the states on the path that can be easily reached from each other. Others [29], [30] compressed all-pairs shortest paths data. These compressions were, however, specific to given search problems (e.g., to a given map) and not portable.

Previous work [31], [32] captured the structure of a search graph by embedding it in a Euclidean space so that Euclidean distance between embedded vertices serves as the heuristic for A*. While Euclidean distance itself is a graph-independent formula, the Euclidean embedding is not and has to be computed on a per-graph basis. In this paper we attempt to automatically extract knowledge of search problems in a problem-independent format which can then be applied to other, similar problems without additional pre-processing.

IV. Our Approach

We first present the base algorithm a which we seek a good initial heuristic for. We then describe our heuristic space $\mathcal{H}$ and how we search for the initial heuristic $h_0^{\min}$ in it.

Henceforth we will use the following notation: $x \sim U(X)$ means that the variable x is uniformly randomly sampled from the set X. Similarly, $X' \sim U^{\not\subset}(X, n)$ denotes an n-element subset $X' \subset X, n \le |X|$ where each element of X' is uniformly randomly drawn from X *without* replacement.

A. The Base Algorithm

In this paper we search for initial heuristics to be used with the seminal LRTA* algorithm [2]. Since its introduction around thirty years ago LRTA* has been the basis for most of the newer and more powerful RTHS algorithms. Thus we hope that the approach we present here will generalize onto contemporary algorithms.

LRTA* with the lookahead of 1 is shown as Algorithm 1. As long as the goal state s_g is not reached it interleaves three steps: (i) generating the local search space $N(s_t)$, (ii) updating the heuristic (line 3) and (iii) moving to the most promising state in the local search space (line 4) with ties broken randomly per move. The mini-min rule used for learning heuristic is a deterministic version of the Bellman optimality equation turned into an assignment operator. Indeed the unique solution to the system of $|S|$ equations $\left\{h(s) = \min_{s' \in N(s)} (c(s,s') + h(s'))\right\}_{s \in S}$ is the perfect heuristic function h^*. Or put another way, the perfect heuristic h^* is the unique fixed point for the heuristic learning/update rule used in line 3.

B. The Space of Initial Heuristics

Given the desiderata of Section II-C we represent heuristics as algebraic expressions. For clarity, simplicity and without loss of generality, we assume that graph vertices are described by pairs of coordinates. Thus the heuristic $h(x_1, y_1, x_2, y_2)$

Algorithm 1: LRTA* with lookahead of 1

```
input : search problem $(S, E, c, s_0, s_g)$, initial heuristic $h_0$
output: solution $(s_0, s_1, \ldots, s_g)$
1 $t \leftarrow 0$
2 while $s_t \neq s_g$ do
3     $h_{t+1}(s_t) \leftarrow \min_{s \in N(s_t)} (c(s_t, s) + h_t(s))$
4     $s_{t+1} \leftarrow \operatorname{argmin}_{s \in N(s_t)} (c(s_t, s) + h_t(s))$
5     $t \leftarrow t + 1$
```

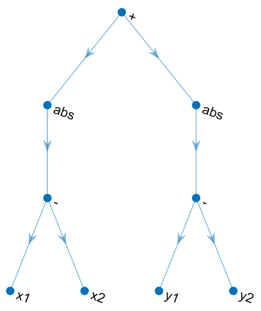

Fig. 1. Manhattan distance as a syntax tree.

estimates path cost between states (x_1, y_1) and (x_2, y_2). Then formulae representing heuristics are defined over variables x_1, y_1, x_2, y_2, numeric constants and various operators. While one could evolve formulae as text strings, most random mutations would lead to syntactically invalid expressions. Thus we represented formulae as syntax trees. Tree leaves are terminal nodes (numeric constants in $\{1, 2, \ldots, 6\}$ and variables x_1, y_1, x_2, y_2) while tree internal nodes are unary operators (−, principal square root, square and absolute value) and binary operators $(\max, \min, +, -, *, /)$. Figure 1 shows a syntax tree for Manhattan distance.

In choosing the set of possible syntax-tree nodes we attempted to balance formula expressiveness and heuristic space size. For instance, preliminary experiments showed that including logarithm does not reliably yield better heuristics.

C. Searching the Space

Since we do not know the specific heuristic function we are searching for we do not have a goal-state test for the space. That excludes standard search methods which require such a goal-state test. Furthermore, the space of heuristics, even if we realistically cap the size of syntax trees, is too large to attempt an exhaustive search. Previous work [11] systematically sampled a space of RTHS algorithms by tabulating ranges of the control parameters defining the algorithms. However, it is not clear how to use their approach here since each heuristic is not naturally represented as a vector of fixed dimension. Additionally, random sampling of the heuristic space reveals that even the baseline human-designed heuristic, Manhattan distance, outperforms virtually all heuristics formed by starting with a random single-node syntax tree and then mutating it $U(\{1, \ldots, 100\})$ times as detailed below. This is in contrast to the previously explored space of RTHS algorithms explored [12] where most randomly sampled algorithms outperformed the baseline (RTA*).

The approach we implement in this work adds guidance to random sampling of the search space, directing it towards more promising heuristics. Specifically, we run a series of simulated evolutions of heuristic functions each with a progressively larger population, more generations and a more accurate but more computationally expensive fitness function. Each evolution is partly seeded with the best heuristic found so far. Our hope is that early evolution runs will quickly bring the population to a more promising area of the heuristic space. Later evolution runs then refine the solution. Additionally, restarting evolution on a regular basis helps escape local minima and plateau that an evolving population can reach. While these ideas are well known in the field of evolutionary computation, their application to finding an initial heuristic function for RTHS is novel, to the best of our knowledge.

A single evolution run is presented as Algorithm 2. The evolution starts with a random population (one of its members set to a seed heuristic h_s, if given) and continues for G generations (unless the quota Q on the number of heuristics evaluated is reached first). A random heuristic is generated by creating a syntax tree of a single random terminal node and then mutating it $U(\{1, \ldots, u\})$ times. On each generation g we select a subset P' of M problems by randomly drawing from the training set P without replacement in line 7. Fitness of each heuristic h is computed in line 9 as reciprocal of 1 plus the natural logarithm of h's suboptimality with the RTHS algorithm a on the problem set P' capped by $\alpha_{\max}$. We then subtract the regularization term $\lambda|h|$ where $|h|$ is the number of nodes in the syntax tree for h. Any heuristic that gives a constant value for all problems in P' gets the fitness of $-\infty$. Note that computing each h's fitness on a random subset $P' \subset P$ greatly accelerates evolution since $M = |P'| \ll |P|$. We form P' randomly on each generation to avoid overfitting h to a specific (small) subset of the training set. The best heuristic found so far is updated in line 12, ties in fitness broken randomly.

The next generation is formed in line 13 by keeping $\lfloor \beta |H| \rfloor$ top-fitness heuristics out of H and replacing the rest with their offspring. Each of the $|H| - \lfloor \beta |H| \rfloor$ offspring is formed by picking a random parent from the $\lfloor \beta |H| \rfloor$ top-fitness heuristics and cloning its syntax tree into the offspring. We then mutate the syntax tree $\lceil 0.1 + |e| \rceil$ times where e is drawn from an exponential distribution with the mean parameter μ. Each mutation selects a random node in the syntax tree and randomly modifies it. A binary operator can be changed to another binary operator or a unary operator with one of the arguments (i.e., a subtree) discarded. A binary node can also be replaced with a terminal node, losing both of its subtrees. An unary operator can also be inserted between a binary operator and one of its arguments. Finally, one of the binary operator's arguments can be dropped and the other can be attached directly to the binary operator's parent in the syntax tree. Similar modifications are applied to unary operators and leaf nodes (omitted for brevity).

Progressive evolution is presented as Algorithm 3. As long

Algorithm 2: Single evolution run `evolve`

```
input : training problem set $P$, sample size $M$, seed heuristic
        $h_s$, population size $N$, number of generations $G$,
        suboptimality cap $\alpha_{\max}$, regularizer $\lambda$, quota $Q$,
        parenting proportion $\beta$, RTHS algorithm $a$, parameter $\mu$
output: best found heuristic $h_{\text{best}}$, heuristics evaluated $q_{\text{spent}}$
1  $q \leftarrow 0$
2  $g \leftarrow 0$
3  $h_{\text{best}} \leftarrow \varnothing$
4  randomly create heuristic population $H$ of size $N$
5  set one of $H$ to the seed heuristic $h_s$
6  while $q < Q$ & $g < G$ do
7      select subset of problems $P' \sim U^{\Phi}(P, M)$
8      for $h \in H$ do
9          $\phi(h) \leftarrow \frac{1}{1+\ln \alpha(a,h,P')} - \lambda|h|$
10     $q \leftarrow q + N$
11     if $\max_{h\in H} \phi(h) > \phi(h_{\text{best}})$ then
12         $h_{\text{best}} \leftarrow \arg\max_{h\in H} \phi(h)$
13     $H \leftarrow \beta H \cup \text{offspring}(\beta H)$
14     $g \leftarrow g + 1$
15 $q_{\text{spent}} \leftarrow q$
```

Algorithm 3: Progressive evolution

```
input : training problem set $P$, sample size $M$, population size
        $N$, number of generations $G$, suboptimality cap $\alpha_{\max}$,
        regularizer $\lambda$, quota $Q$, parenting proportion $\beta$, RTHS
        algorithm $a$, mean parameter $\mu$, scale factor $\eta$, scale
        factor $\omega$, maximum stall $s_{\max}$, parameter $\mu$
output: heuristic $h_{\text{historic best}}$
1  $q \leftarrow 0$
2  $s \leftarrow 0$
3  $h_{\text{historic best}} \leftarrow \varnothing$
4  while $q < Q$ do
5      $h_{\text{evolved}}, q_{\text{spent}} \leftarrow$
       evolve$(h_{\text{historic best}}, P, M, N, G, \alpha_{\max}, \lambda, Q - q, \beta, a, \mu)$
6      $q \leftarrow q + q_{\text{spent}}$
7      if $\alpha(a, h_{\text{evolved}}, P) < \omega\alpha(a, h_{\text{historic best}}, P)$ then
8          $h_{\text{historic best}} \leftarrow h_{\text{evolved}}$
9          $s \leftarrow 0$
10     else
11         $s \leftarrow s + 1$
12         if $s > s_{\max}$ then
13             $N \leftarrow \eta N$
14             $G \leftarrow \eta G$
15             $M \leftarrow \min\{\eta M, |P|\}$
16             $\lambda \leftarrow \eta\lambda$
17             $\alpha_{\max} \leftarrow \eta\alpha_{\max}$
18             $s \leftarrow 0$
```

as the quota Q of heuristics is not exhausted progressive evolution repeatedly executes single evolutions runs in line 5. The evolved heuristic h_{evolved} is then evaluated on the full training set P (using a fixed suboptimality cap $\alpha'_{\max}$). If it is better than $h_{\text{historic best}}$ by at least ω (line 7) then the latter is updated in line 8. Otherwise we consider the evolution to stall and increase the stall duration s. If the stall has been going on for long enough ($s > s_{\max}$ in line 12) then we scale up the parameters defining a single evolution run by the factor $\eta > 1$ each in lines 13-17. The next evolution run has one of its population members set to $h_{\text{historic best}}$ with the others formed randomly.

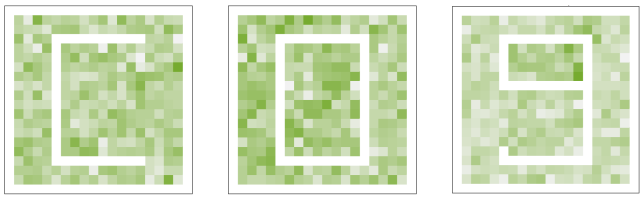
Fig. 2. Our grid-based pathfinding maps.

V. Empirical Evaluation

In this section we set up our problem domain, describe the heuristic-evolution experiments, analyze how each evolved heuristic appears to compress characteristics of a grid map and analyze portability of the evolved heuristics.

A. Grid-based Pathfinding

In the long-running tradition of RTHS field we evaluate our approach on grid-based pathfinding. To explore whether our evolved heuristics can capture specific characteristics of a map such as presence of heuristic depressions we need maps that clearly display such characteristics. The typically used video-game maps [33] often combine diverse characteristics in a single map (e.g., one can often find open areas and small rooms in the same map). Furthermore, we wanted the maps to be small enough so that the knowledge a heuristic is capturing can be easily displayed and analyzed in the paper. Thus we built our own three maps (Figure 2), 20×20 grid cells each. Pure white cells represent obstacles, green cells are open.

To avoid evolving heuristics that exploit low-level errors of floating-point arithmetics in a numeric library, we considered the maps to be 4-connected with each cell having up to four neighbours in the cardinal directions. Each move costs 1. This allowed us to switch from floating-point arithmetic and its approximation to the irrational cost of the diagonal moves to exact integer-valued arithmetic. The baseline heuristic between states (x_1, y_1) and (x_2, y_2) is thus Manhattan distance (MD) $h(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$. The three maps vary in their difficulty for LRTA* with MD as we will see below.

To compute the initial heuristic for a state (x_1, y_1), the heuristic formula is evaluated on the state coordinates (x_1, y_1) and the goal coordinates (x_2, y_2) and calculated as a floating-point number, possibly complex. Any imaginary parts are then dropped and the rest is rounded to an integer. If the result is infinity then it is replaced by the largest signed integer representable in 64 bits (i.e., $2^{63} \approx 9.2 \times 10^{18}$). If the formula's expression is undefined for given inputs (e.g., the principal square root of -1) then it is replaced by 0. Note that the initial heuristic formula is used only to fill in a heuristic table which is then passed to the RTHS agent. Any updates to the heuristic the agent performs while traversing the search graph are done with the table.

We formed three sets of random problems. The set P_C^{10K} had 10 thousand pathfinding problems on the map *C*. To form the set we generated 100 unique random goals and for each goal selected 100 random start states, making sure the start and goal states of any problem are distinct and the goal is indeed reachable from the start. Similarly, we formed the problem sets P_O^{10K}, P_G^{10K} for the other two maps. In Figure 2 each cell's shade of green indicates the number of start states falling on that cell.

B. Evolving Heuristics

To evaluate our method of evolving heuristics we ran four folds for each of the three problems sets $P_C^{10K}, P_O^{10K}, P_G^{10K}$. For each fold we selected 75 of the problem set's 100 goals for training and the remaining 25 goals for testing. Thus the training partition of the problem set contained $75 \times 100 = 7500$ problems while the test partition contained the other $25 \times 100 = 2500$ problems. Since all 100 goals were unique for each map, the partitioning of each P^{10K} into training and testing problem sets was disjoint with no shared problems. The first fold used the first 25 goals (and the associated 2500 problems) for testing, the second fold used the next 25 goals and so on.

On each fold we ran progressive evolution (Algorithm 3) on the fold's 7500 training problems. We used LRTA* with the lookahead of 1 (Algorithm 1) as the algorithm a. Other hyperparameters are listed in Table II. The resulting, evolved heuristic $h_{\text{historic best}}$, was then evaluated on the fold's 2500 test problems. As there were four folds, progressive evolution was run four times. The four evolved heuristics for map C are shown in Table I, algebraically simplified for the presentation. For each fold the table also lists the test suboptimality of the evolved heuristic as well as the baseline heuristic (Manhattan distance). For all test sets the suboptimality cap was set to ∞ so the test suboptimality is the mean of uncapped solution suboptimality over the 2500 test problems for that fold.

Due to space limitations, we do not show the other two tables for maps O and G. Instead, Table III shows test suboptimality of evolved heuristics averaged over the four folds for all three maps. Evolved heuristics had better test suboptimality averaged over the four folds than Manhattan distance. The sole exception was map O which is simple enough that Manhattan distance does very well: 1.44 ± 0.088.

C. An Insight Into Evolved Heuristics

Unlike deep neural networks or large pattern databases, heuristics represented as short algebraic expressions have a potential to be more human-interpretable. Consider, for instance, heuristic $h_C = \sqrt{\max\left\{2, \left(\frac{x_2}{x_1}\right)^{16}\right\}} + |y_1 - y_2| + \min\{\sqrt{x_1 - 1}, 6\}$ evolved in fold 3 on map C (Table I). The heuristic outperforms Manhattan distance on average over test problems of that fold: 2.22 versus 5.22. Why?

The answer lies with the fact that the evolved heuristic prioritizes the difference in the y coordinates of the agent and the goal over the difference in the x coordinates. Consequently, the agent is less likely to be lured into a heuristic depression by

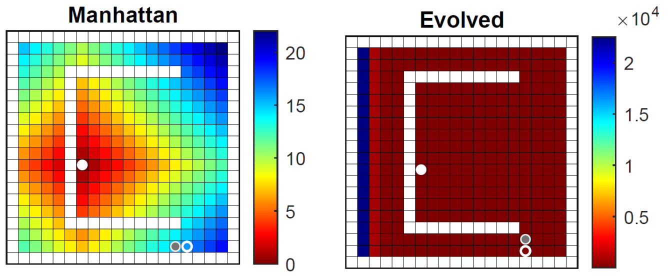

Fig. 3. A problem on map *C*. The colors show initial heuristic values. The goal is a white dot. The start is a white circle.

moving horizontally towards the goal. To illustrate, consider the specific problem in Figure 3. With Manhattan distance the agent can step to the left (the gray dot in the figure), led by the decrease in $|x_1 - x_2|$. Doing so puts a wall between it and the goal and causes it to wander around until it gets back inside the letter C.

With the evolved heuristic the difference in x coordinates is less important than the difference in the y coordinates and the agent heads up, minimizing the $|y_1 - y_2|$ term. In this particular problem, doing so guides it up towards the goal. Once it matches the goal's y coordinate it will step left towards the goal, guided by minimizing $\sqrt{x_1 - 1}$. The important point is that focusing on equalizing the y coordinate and effectively ignoring the x coordinates in the start state does not put a wall between the agent and its goal.

Naturally for some other problems prioritizing y coordinates over x coordinates can be misguiding. Evolution simply picks up on what is more beneficial on average for a given map. Map C has its largest heuristic depression oriented horizontally which means that downplaying the difference in x coordinates can likely be of benefit more often than not.

D. Portability of Evolved Heuristics

The evolved heuristics captured knowledge of training problems and generalized it onto test problems unseen during training. In this section we look at how general the evolved heuristics are. To do so we generated an additional set of 100 thousand problems (100 goals, 1000 starts/goal) for each of the three maps: P_C^{100K}, P_O^{100K}, P_G^{100K}. We then ran LRTA* with the best-of-the-folds evolved heuristic on the problem sets. The resulting suboptimality (mean ± standard error of the mean; the suboptimality cap was set to 10^3) is in Table IV, best suboptimality per problem set is in bold.

The evolved heuristics work best on the maps they were trained for. They are too map-specific to outperform Manhattan distance on other maps. We believe this is due to the fact that evolved heuristics exploit peculiarities of their training maps whereas the Manhattan distance does not and thus ends up being a jack of all trades, master of none, so to speak.

VI. Current Shortcomings & Open Questions

The heuristics we evolved in this paper provide a better guidance than Manhattan distance because they appear to fit to some characteristics shared by a class of pathfinding problems.

TABLE I
EVOLVED HEURISTICS FOR MAP C AND THEIR TEST SUBOPTIMALITY COMPARED TO MANHATTAN DISTANCE.

Fold	Evolved heuristic	Test suboptimality	MD test suboptimality
1	$\|y_1 - y_2\| - \min\{x_1, x_2\}^2$	**4.89**	8.88
2	$\sqrt{5\,\|x_2 - x_1\| + 6} - \min\{x_1, x_2\}^2$	**6.73**	7.05
3	$\sqrt{\max\left\{2, \left(\frac{x_2}{x_1}\right)^{16}\right\} + \|y_1 - y_2\| + \min\left\{\sqrt{x_1 - 1}, 6\right\}}$	**2.22**	5.22
4	$\|y_1 - y_2\|$	**3.81**	3.92

TABLE II
HYPERPARAMETERS USED.

Parameter	**Value**
initial sample size M	100
initial population size N	40
initial number of generations G	10
initial regularizer λ	0.0001
initial suboptimality cap $\alpha_{\max}$	10
number of heuristics evaluated Q	5×10^4
parenting proportion β	0.1
RTHS algorithm a	LRTA*
scale factor η	1.5
scale factor ω	0.95
maximum stall $s_{\max}$	1
training set cap $\alpha'_{\max}$ (line 7)	100
exponential distribution mean parameter μ	5
maximum number of initial mutations u	100

TABLE III
TEST SUBOPTIMALITY OF EVOLVED HEURISTICS AVERAGED OVER THE FOUR FOLDS. STANDARD DEVIATIONS ARE ALSO LISTED.

Map	Test suboptimality (evolved)	Test suboptimality (MD)
C	$\mathbf{4.41 \pm 1.893}$	6.27 ± 2.165
O	1.60 ± 0.338	$\mathbf{1.44 \pm 0.088}$
G	$\mathbf{3.01 \pm 0.339}$	4.68 ± 0.997

Thus a natural open question is how diverse such a class can be while remaining amenable to our approach. Can it be that once a class of search problems is sufficiently diverse its characteristics can no longer be captured by an algebraic formula and thus a default heuristic such as MD can no longer be beaten? An investigation into this can yield a theoretical result similar to A*'s search cost being unbeatable unless problem-specific information is taken advantage of. Perhaps linking properties of search problems [34] and properties of heuristics can be helpful here.

Evolving heuristics can be expensive since each candidate has to be evaluated by running it with an RTHS algorithm. We addressed this problem by estimating the actual fitness via fitness on a smaller subset. The size of the subset was then incrementally increased in the progressive evolution, making the fitness function estimate progressively more accurately. Still, we have not been able to reliably outperform MD on actual video-game maps. The challenge is that video-game maps are too difficult for the basic LRTA* making computing heuristic fitness too expensive to allow for a sufficiently large evolution. Future work will investigate replacing our sample fitness with even a faster-to-compute estimate such as a neural net or the k nearest neighbours.

Evolving a heuristic takes time while MD is available at the outset. On the other hand using an evolved heuristic instead of MD reduces search time on many problems. Thus, future work will consider amortizing heuristic evolution cost over a stream of problems. It will also be of interest to automatically select a heuristic on a per-problem [35] or even per-step basis. The latter can be viewed as global learning wherein, instead of updating the heuristic locally via mini-min, the agent globally changes all heuristic values by loading a new initial heuristic when it realizes that it is not making sufficient progress with the current heuristic. A special case of this is using a generic heuristic such as MD as a fall back (a la [36]).

We used a basic RTHS algorithm, LRTA*, throughout the paper. Future work will apply our approach to contemporary real-time and non-real-time search algorithms. This should help scaling our approach to actual video games which are too difficult for the basic LRTA*. One can also co-search an algorithm space [11] and a heuristic space simultaneously.

Future work will also consider maps that can change dynamically [37]. This is common in video games either due to multi-agent pathfinding or to terrain changes (e.g., the player builds a bunker to block an entrance to their base). Recently RTHS algorithms have been used in multi-agent pathfinding albeit with a standard heuristic [38]. To what extent can our approach of evolving heuristics specifically to a map be used on dynamic maps and multi-agent pathfinding? Another challenging extension to the pathfinding model is terrain with different traversal costs [22].

TABLE IV
PORTABILITY OF EVOLVED HEURISTICS.

	P_C^{100K}	P_O^{100K}	P_G^{100K}
MD	6.14 ± 0.033	1.45 ± 0.004	4.60 ± 0.024
h_C	$\mathbf{2.96 \pm 0.019}$	1.69 ± 0.007	224.93 ± 1.280
h_O	19.44 ± 0.113	$\mathbf{1.17 \pm 0.002}$	11.05 ± 0.068
h_G	359.58 ± 1.514	1.41 ± 0.005	$\mathbf{2.18 \pm 0.013}$

VII. Conclusions

This paper presents the first attempt to automatically build compact, human-readable initial heuristics for real-time heuristic search. We represented each heuristic as an algebraic expression and demonstrated that progressive evolution can find heuristics with better LRTA* performance than the standard Manhattan distance on toy-sized grid maps. The evolved heuristics are human-readable and portable to problems not seen during evolution (although only on the same map).

The approach promises a way to automatically take advantage of properties shared by a class of heuristic problems. From a machine-learning perspective, our work splits heuristic learning between off-line learning over multiple generations and on-line learning during the agent's lifetime. In A-life terms this gives an agent innate knowledge to survive after birth yet allows it to improve the knowledge via life experience making the agent more adaptable to novel/changing environments.

Acknowledgments

The idea of automatically finding closed-form representations for initial heuristics came out of discussions between the author and Valeriy Bulitko. Jonathan Schaeffer suggested limiting numeric constants to a small pre-defined set.

References

[1] S. Koenig, "Agent-centered search," *Artificial Intelligence Magazine*, vol. 22, no. 4, pp. 109–132, 2001.

[2] R. Korf, "Real-time heuristic search," *Artificial Intelligence*, vol. 42, no. 2–3, pp. 189–211, 1990.

[3] N. Sturtevant and V. Bulitko, "Scrubbing during learning in real-time heuristic search," *Journal of Artificial Intelligence Research*, vol. 57, pp. 307–343, 2016.

[4] D. Ackley and M. Littman, "Interactions between learning and evolution," *Artificial life II*, vol. 10, pp. 487–509, 1991.

[5] S. Russell and E. Wefald, *Do the Right Thing: Studies in Limited Rationality*. MIT Press, 1991.

[6] V. Bulitko, "Effects of self-knowledge: Once bitten twice shy," in *Proceedings of the Experimental AI in Games (EXAG) Workshop at the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2017, pp. 26–33.

[7] V. Bulitko and K. Doucet, "Anxious learning in real-time heuristic search," in *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*, 2018, pp. 393–396.

[8] D. Kahneman, *Thinking, fast and slow*. Farrar, Straus and Giroux, 2011.

[9] S. Koenig, "A comparison of fast search methods for real-time situated agents," in *Proceedings of the International conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2004, pp. 864–871.

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

[11] V. Bulitko, "Evolving real-time heuristic search algorithms," in *Proceedings of the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*, 2016, pp. 108–115.

[12] ——, "Searching for real-time search algorithms," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2016, pp. 121–122.

[13] S. Koenig and M. Likhachev, "Real-time adaptive A*," in *Proceedings of the International conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2006, pp. 281–288.

[14] N. R. Sturtevant and V. Bulitko, "Reaching the goal in real-time heuristic search: Scrubbing behavior is unavoidable," in *Proceedings of the Symposium on Combinatorial Search (SoCS)*, 2014, pp. 166–174.

[15] N. Rivera, J. A. Baier, and C. Hernández, "Incorporating weights into real-time heuristic search," *Artificial Intelligence*, vol. 225, pp. 1–23, 2015. [Online]. Available: http://dx.doi.org/10.1016/j.artint.2015.03.008

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

[17] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical pathfinding," *Journal of game development*, vol. 1, no. 1, pp. 7–28, 2004.

[18] V. Bulitko, N. Sturtevant, J. Lu, and T. Yau, "Graph abstraction in real-time heuristic search," *Journal of Artificial Intelligence Research*, vol. 30, pp. 51–100, 2007.

[19] N. R. Sturtevant, "Memory-efficient abstractions for pathfinding," in *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2007, pp. 31–36.

[20] V. Bulitko and Y. Björnsson, "kNN LRTA*: Simple subgoaling for real-time search," in *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2009, pp. 2–7.

[21] R. Lawrence and V. Bulitko, "Taking learning out of real-time heuristic search for video-game pathfinding," in *Proceedings of Australasian Joint Conference on Artificial Intelligence*, 2010, pp. 405–414.

[22] N. R. Sturtevant, D. Sigurdson, B. Taylor, and T. Gibson, "Pathfinding and abstraction with dynamic terrain costs," in *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2019, pp. 80–86.

[23] Y. Björnsson and K. Halldórsson, "Improved heuristics for optimal pathfinding on game maps," in *Proceedings of Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*, 2006, pp. 9–14.

[24] N. R. Sturtevant, A. Felner, M. Barrer, J. Schaeffer, and N. Burch, "Memory-based heuristics for explicit state spaces," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2009, pp. 609–614.

[25] A. Sadikov and I. Bratko, "Pessimistic heuristics beat optimistic ones in real-time search," in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2006, p. 148–152.

[26] M. Luštrek and V. Bulitko, "Thinking too much: Pathology in pathfinding," in *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, 2008, pp. 899–900.

[27] J. Culberson and J. Schaeffer, "Pattern Databases," *Computational Intelligence*, vol. 14, no. 3, pp. 318–334, 1998.

[28] A. Felner, R. E. Korf, R. Meshulam, and R. C. Holte, "Compressed pattern databases," *Journal of Artificial Intelligence Research (JAIR)*, vol. 30, pp. 213–247, 2007.

[29] A. Botea, "Ultra-fast optimal pathfinding without runtime search," in *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2011, pp. 122–127.

[30] A. Botea and D. Harabor, "Path planning with compressed all-pairs shortest paths data," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 2013, pp. 293–297.

[31] D. C. F. Rayner, M. H. Bowling, and N. R. Sturtevant, "Euclidean heuristic optimization," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2011, pp. 81–86.

[32] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, and T. S. Kumar, "The FastMap algorithm for shortest path computations," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2018, pp. 1427–1433.

[33] N. R. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012.

[34] D. A. Huntley and V. Bulitko, "Search-space characterization for real-time heuristic search," *CoRR*, vol. abs/1308.3309, 2013. [Online]. Available: http://arxiv.org/abs/1308.3309

[35] D. Sigurdson and V. Bulitko, "Deep learning for real-time heuristic search algorithm selection," in *Proceedings of the Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*, 2017, pp. 108–114.

[36] F. Muñoz, M. Fadic, C. Hernández, and J. A. Baier, "A neural network for decision making in real-time heuristic search," in *Proceedings of the Annual Symposium on Combinatorial Search (SoCS)*, 2018, pp. 173–177.

[37] M. Bono, A. E. Gerevini, D. D. Harabor, and P. J. Stuckey, "Path planning with CPD heuristics," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2019, pp. 1199–1205.

[38] D. Sigurdson, V. Bulitko, W. Yeoh, C. Hernández, and S. Koenig, "Multi-agent pathfinding with real-time heuristic search," in *Proceedings of the Conference on Computational Intelligence and Games (CIG)*, 2018, pp. 173–180.