Online Learning of Search Heuristics

Michael Fink
Center for Neural Computation
The Hebrew University of Jerusalem
Jerusalem, Israel 91904

Abstract

In this paper we learn heuristic functions that efficiently find the shortest path between two nodes in a graph. We rely on the fact that often, several elementary admissible heuristics might be provided, either by human designers or from formal domain abstractions. These simple heuristics are traditionally composed into a new admissible heuristic by selecting the highest scoring elementary heuristic in each distance evaluation. We suggest that learning a weighted sum over the elementary heuristics can often generate a heuristic with higher dominance than the heuristic defined by the highest score selection. The weights within our composite heuristic are trained in an online manner using nodes to which the true distance has already been revealed during previous search stages. Several experiments demonstrate that the proposed method typically finds the optimal path while significantly reducing the search complexity. Our theoretical analysis describes conditions under which finding the shortest path can be guaranteed.

1 Introduction

Finding the shortest path between two nodes in a graph is a problem which emerges in many domains ranging from train scheduling (Schulz *et al.*, 1999) to computer graphics (Omer and Werman, 2006). However, the time complexity of the popular Dijkstra algorithm is intolerable if a quick answer is required while searching large graphs. This led researches to suggest algorithms, such as A^*, (Hart *et al.*, 1968), which introduce a heuristic factor into the search process. In addition to the standard search problem variables (graph, starting node and goal node), the A^* algorithm receives as input a heuristic function, h, which is used to evaluate the remaining distance from each visited node to the goal node. Adding this heuristic score to the nodes stored in Dijkstra's priority queue can effectively prune down the number of nodes visited during the search.

The A^* algorithm guarantees that the shortest path is returned, if the heuristic function h is *admissible*, namely it never overestimates the true distance from a certain node to the goal. In addition, the heuristic *dominance* characteristic provides that during the search, an admissible heuristic h which consistently generates higher estimations than the admissible heuristic h', will never visit more nodes than h' (Russell and Norvig, 2003).

In this paper we assume that several simple admissible heuristics are provided, either by a human designer or from formal domain abstractions (Minsky, 1963; Pearl, 1984; Holte and Hernadvolgyi, 2004). By taking the maximum value over these elementary heuristics a new admissible heuristic can be generated. We term this the *maximum value* approach. However, although the maximum value of many elementary heuristics is guaranteed to be an admissible heuristic, it might not be highly dominant. To see this, let us assume a simplified n-dimensional binary search space $V = \{0,1\}^n$, in which the true distance between nodes $v \in V$ and $t \in V$ is $\sum_{i=1}^{n} |v_i - t_i|$. If each domain abstraction provides the binary distance of a single dimension, then the maximum over all abstractions will be no larger than 1 (while the true distance might be in the order of n). This paper suggests an alternative method for automating the heuristic composition process. Rather than relying on taking the maximal value over the available abstractions, we utilize state-of-the-art online learning mechanisms in order find an appropriate weight for summing the elementary heuristic values. We suggest that when the different elementary heuristics capture different aspects of the distance underlying the search space, weighting the individual abstractions is preferential to simply selecting the highest scoring one.

The proposed approach is especially appealing in settings where manually designing a heuristic function might not be possible. For example, imagine a large scale distributed communication network used by military vehicles in which every node can transmit, receive or relay communications

from neighboring nodes. In this ad-hoc graph, the goal is to transmit communications in real time through the least energy consuming path. Since the graph might be continuously changing as the forces progress a static heuristic function might be sub-optimal[1].

Our setting assumes that a sequence of related search tasks must be performed using an A^* mechanism. At the end of each search task the currently held composite heuristic receives feedback indicating where its estimates have maximally deviated from the true distances. Using this feedback the parameters of the heuristic function are updated so it can better capture the characteristics of the distance underlying the specific sequence of search tasks. Our setting is related to other repeated graph search settings (Korf, 1990; Culberson and Schaeffer, 1996; Edelkamp and Eckerle, 1997; Koenig *et al.*, 2004), however, while these methods rely on memorization for transferring heuristic knowledge, our method relies on the machine learning notion of generalization. This approach is inspired by previous attempts to learn the specific characteristics of large search spaces in order to facilitate searching procedures (Boyan and Moore, 2000). The novel component in our work is in adapting this approach to the shortest path problem and in our attempt to derive formal optimality guarantees on the the returned paths.

The feedback signal we require for training the heuristic function h, could naturally emerge as part of the search process. For example, in the vehicle communication system, the shortest path search procedure must be performed efficiently. However, once the goal node was found in *real time*, the system can derive the evaluation feedback by performing an exact search *off line*. This reverse search can start from the goal node and utilize the maximum value heuristic in order to calculate the true distances to all of the evaluations the heuristic h performed during the *real time* search. Unlike traditional online supervised learning settings in which labels require an external teacher, training in search problems has the elegant property of being able to rely on exact search mechanisms for supervision.

2 Problem Setting

Let $G = (V, E)$ be a graph in which each edge $(v \in V, v' \in V) \in E$ is associated with a positive cost $c(v, v') \geq 0$. We define a *shortest path search problem* by the triplet (G, s, g) where $s \in V$ is the source node and $g \in V$ is the goal node. Let $(G_1, s_1, g_1) \ldots (G_T, s_T, g_T)$ be an online sequence of shortest path search problems. At each round $1 \leq t \leq T$, the algorithm receives search problem (G_t, s_t, g_t) and must produce the edges of the shortest path p_t from s_t to g_t.

[1]Throughout the paper we assume that dynamic graph changes occur in a longer time constant than that of the search process.

Our setting follows the A^* framework and assumes that the number of nodes visited during the search process must be minimized using a heuristic function $h : V \times V \rightarrow \mathbb{R}$. This heuristic function guides the search by prioritizing which of the nodes in the search frontier will be expanded next. The priority score $f(v) = k(s, v) + h(v, g)$ is the sum of the distance $k(s, v)$, required to reach node v from node s, and of the heuristic contribution $h(v, g)$ estimating the remaining distance needed to reach the goal node g from node v. Unlike the traditional A^* setting in which the heuristic function h is manually designed, and assumed to be known in advance, our setting assumes that h is *not* known a-priori and must therefore be learned online during the T search rounds.

Our setting focuses on heuristics over the *search-space* V, which are induced by linear regression functions in a related $\mathbb{R}^n$ *learning-space*,

$$h(v, v') = \mathbb{T}(\mathbf{w} \cdot \phi(v, v')) \ . \tag{1}$$

It is assumed that a function $\phi : V \times V \rightarrow \mathbb{R}^n$ is provided, which receives a pair of nodes from the search space (v, v') and returns an n dimensional feature mapping in the learning-space. Each feature ϕ_i is an elementary heuristic, typically a simplification generated by ignoring some of the domain constraints. The regression function multiplies the features of ϕ by a weight vector $\mathbf{w} \in \mathbb{R}^n$ and then applies some reversible non-decreasing function, $\mathbb{T} : \mathbb{R} \rightarrow \mathbb{R}$, mapping distance values from the learning-space, back to distances in the search-space. We will later rely on the assumption that this transformation must maintain that as a certain distance in the learning-space goes to 0, the analog distance in the search-space goes to 0 as well. Although somewhat limited, we demonstrate that this family of heuristics is not as meager as it initially seems.

At each round in our online setting an A^* search process is performed using the current heuristic function h. During this search the algorithm visits a set of nodes, we denote as M_t. When the search is concluded, the shortest path p_t must be returned. In order to tune the vector of regression parameters $\mathbf{w}$, it is assumed that after the shortest path is found the algorithm receives a feedback signal. This feedback includes the node $v_t \in M_t$ on which the current heuristic estimation had the maximal (learning-space) deviation from $y_v = \mathbb{T}^{-1}(d(v, g))$, or more formally,

$$v_t = \operatorname*{argmax}_{v \in M_t} \ |y_v - \mathbf{w}_t \cdot \phi(v, g_t)| \ .$$

In addition to the maximally deviating node, the feedback also includes the true distance $y_t = \mathbb{T}^{-1}(d(v_t, g))$. Using this feedback the algorithm can update the weights of the heuristic function h in order to improve its performance in the subsequent rounds.

For concreteness let us review an example using the map in Figure 1, depicting the road grid and the shopping malls

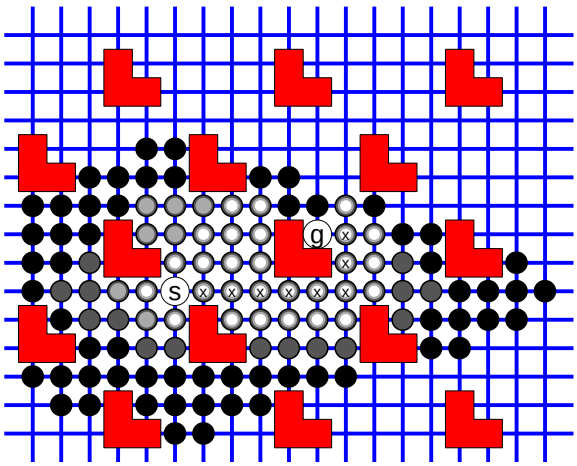

Figure 1: A graph of the road grid in Anytown, USA. The L-shaped structures represent shopping malls. Nodes on the shortest path from s to g are indicated by $\times$. Notice the increasing heuristic dominance between: Dijkstra (Black), Block Distance (Dark Gray), Optimal Admissible (Light Gray) and Online Learning to Search (White).

in Anytown, USA. Let us assume that the search task at round t emerges from a car driver in position s_t wishing to receive the shortest path to a goal destination g_t. This degenerate example presents the same graph in all search rounds ($\forall_t G_t = G$). In many regions, municipal policy, highway layout or terrain constraints cause the traffic flow in a certain direction (say East-West) to be significantly faster than in other directions (North-South). In Anytown, the cost of driving from any intersection v one block West to intersection v' is $c(v, v') = 1$, while the cost of driving from intersection v one block North to intersection v'' is $c(v, v'') = 3$. Using this information, the length of the shortest path (indicated by $\times$-s) from intersection $s = (6, 6)$ to intersection $g = (11, 8)$ can be calculated as: $7 \times 1 + 2 \times 3 = 13$. An appropriate two dimensional feature mapping for this example is, $\phi_1(v, v') = |v_x - v'_x|$ and $\phi_2(v, v') = |v_y - v'_y|$, where v_x and v_y are the coordinates of intersection v on the road grid (in this example $\mathbb{T}(x) = x$). In the following section we propose an algorithm which learns during an online sequence of searches a weight vector $\mathbf{w}$ aimed at maximizing the tradeoff between admissibility and efficiency. Figure 1 depicts the nodes visited by four algorithms: Dijkstra's Shortest Path Algorithm $\mathbf{w} = (0, 0)$, Block Distance $\mathbf{w} = (1, 1)$, Optimal Admissible $\mathbf{w} = (1, 3)$ and our Online Learning to Search Algorithm. Table 1 presents the path length and the number of nodes visited by the four algorithms averaged over a random selection of 100 start nodes and goal nodes. Our algorithm learns using feedback received at the end of each round, which contains the true distance for the node in the current search that the existing heuristic function maximally deviated from. For example, if at the current round the existing heuristic was defined by $\mathbf{w}_t = (0, 0)$, the maximally deviating node would be node $(2, 2)$ which has an actual distance of 29 to the goal (while $\mathbf{w}_t \cdot \phi((2, 2), (11, 8))$ is 0). Thus, the feedback for training will be $((2, 2), 29)$.

Table 1: Average path length and average visited nodes.

Algorithm	length	# visited
Dijkstra: $\mathbf{w}$=(0,0)	24.47	159.71
Block Distance: $\mathbf{w}$=(1,1)	24.47	103.88
Optimal Admissible: $\mathbf{w}$=(3,1)	24.47	62.01
Online Learning to Search	24.49	44.32

The challenge posed by our setting is to characterize the conditions under which the heuristic function learned by round t is both:

1. admissible ($\forall_v h(v, g_t) \leq d(v, g_t)$) so that the optimality of the returned path can be guaranteed

2. maximally dominant ($\forall_{h',v} h(v, g_t) \geq h'(v, g_t)$), so that the search process is maximally reduced

3 The Online Learning to Search Algorithm

We now describe the learning algorithm aimed at acquiring a heuristic evaluation function during the online search queries. As stated above we assume that a relevant feature mapping $\phi : V \times V \rightarrow \mathbb{R}^n$, is provided and that our task is to learn a weight vector $\mathbf{w} \in \mathbb{R}^n$, characterizing a heuristic function $h(v, v') = \mathbb{T}(\mathbf{w} \cdot \phi(v, v'))$. We would like this heuristic function to efficiently reduce the number of visited nodes during the search while maintaining admissibility, so that finding an optimal path could be guaranteed.

The proposed method relies on the linear regression algorithms described within the online Passive Aggressive framework (Crammer *et al.*, 2006). For clarity of presentation we focus on adapting the simplest mechanism within the Passive-Aggressive framework. This regression mechanism assumes that the family of learned heuristics has the capacity to approximate the true distances $d(v, v')$ up to a constant, ϵ.

Recall that on every round, our algorithm performs an A^* search using the currently held heuristic function. Once concluding this search, the algorithm receives as feedback the node $v_t \in M_t$ on which the current heuristic estimation had the maximal deviation in the learning-space. Thus, the instance used for training in the learning-space is $\phi(v_t, g_t)$ (abbreviate $\phi(t)$) and the target value is $y_t = \mathbb{T}^{-1}(d(v_t, g_t))$. We will later address such pairs $(\phi(t), y_t)$, as our training examples. Our proposed algorithm relies on

the ϵ-insensitive hinge loss function:

$$l_\epsilon(\mathbf{w};(\phi(t),y_t)) = \begin{cases} 0 & |\mathbf{w}\cdot\phi(t)y_t| \leq \epsilon \\ |\mathbf{w}\cdot\phi(t)y_t| - \epsilon & \text{otherwise} \end{cases}$$

where $\epsilon \geq 0$ is a learning feasibility parameter controlling the sensitivity to regression errors. This loss is zero when the predicted target deviates from the true target by less than ϵ and otherwise grows linearly with $|\mathbf{w}\cdot\phi(t)y_t|$.

Our algorithm is initialized by setting $\mathbf{w}_1$ to $(0,\ldots,0)$. At the end of each round, this weight vector is updated to be,

$$\mathbf{w_{t+1}} = \operatorname*{argmin}_{\mathbf{w}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{w}-\mathbf{w_t}\| \quad \text{s.t.} \quad l_\epsilon\big(\mathbf{w};(\phi(t),y_t)\big) = 0 \ .$$

The set $\{\mathbf{w}\in\mathbb{R}^n : l_\epsilon(\mathbf{w};(\phi(t),y_t)) = 0\}$ is a hyper-slab of width 2ϵ. The rational behind this update rule is to perform the minimal adjustment to the present weight vector that makes it accurately predict the target value of round t. Geometrically, $\mathbf{w_t}$ is projected onto the ϵ-insensitive hyper-slab at the end of every round. Using the following three definitions,

$$\begin{aligned} \delta_t(\mathbf{w}) &= y_t - \mathbf{w}\cdot\phi(t) \\ l_{\mathbf{w}_t} &= l_\epsilon(\mathbf{w_t};(\phi(t),y_t)) \\ \tau_t &= \frac{l_{\mathbf{w}_t}}{\|\phi(t)\|^2} \end{aligned}$$

the update rule can be restated by the closed form solution,

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \mathrm{sgn}(\delta_t(\mathbf{w_t}))\ \tau_t\ \phi(t) \ .$$

A summary of the Online Learning to Search Algorithm is presented in Figure 2. It should be noted that (Crammer *et al.*, 2006) provide modifications of the update rule which are more resistant to noisy data and evaluation outliers. The essential change is constraining the magnitude of τ, so that the update steps are less aggressive. In addition, generalizations of this update rule exist for settings where the feedback signal provided at each round, includes *all* the true distances rather than just the most deviant one (Crammer and Singer, 2003). One additional modification might be appropriate for applications in which path optimality is essential. If this is the case we might aim at evaluating $\alpha d(v,v')$ rather than $d(v,v')$, where $0 \leq \alpha \leq 1$, is a parameter controlling the trade-off between path optimality and computational efficiency.

4 Representations in Learning Space

Although, the family of heuristic functions parameterized by Eq. (1) relies on linear regression functions, it nevertheless has the capacity to characterize several interesting search spaces. This will be demonstrated by providing three realization of the feature mapping ϕ and an appropriate reversible non-decreasing transfer function $\mathbb{T}$:

INPUT: $\phi(v,v')$ feature mapping
$\mathbb{T}$ learn-space to search-space transformation
ϵ learning feasibility parameter
INITIALIZE: $\mathbf{w}_1 \leftarrow \mathbf{0}$
For $t = 1,2,\ldots$
define current heuristic $h(v,v') = \mathbb{T}(\mathbf{w}_t\cdot\phi(v,v'))$
receive search problem (G_t,s_t,g_t)
provide path $(p_t,M_t) \leftarrow A^*(G_t,s_t,g_t,h)$
receive $v_t = \operatorname*{argmax}_{v\in M_t} |y_v - \mathbf{w_t}\cdot\phi(v,g_t)|$
where $y_v = \mathbb{T}^{-1}(d(v,g_t))$
set $l_{\mathbf{w}_t} \leftarrow [|y_t - \mathbf{w_t}\phi(t)| - \epsilon]_+$
If $l_{\mathbf{w}_t} > 0$
set: $\tau_t \leftarrow \frac{l_{\mathbf{w}_t}}{\|\phi(t)\|^2}$
update: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathrm{sign}(y_t - \mathbf{w}_t\phi(t))\ \tau_t\phi(t)$

Figure 2: The online learning to search algorithm.

1. **Weighted Block distance** is formally defined as, $\phi_i(v,v') = |v_i - v'_i|$, where ϕ_i, indicates the ith output feature value of the function ϕ. The learned weights over the features (coordinates) express the degree of importance each dimension has in determining the total distance. For weighted block distance the identity transfer function $\mathbb{T}(x) = \mathbb{T}^{-1}(x) = x$, is appropriate. We will later focus our analysis on this type of representation.

2. **Weighted Euclidean distance** can occasionally capture the search-space better than the weighted block distance. Learning this weighted distance could be cast as a linear regression task by defining $\phi_i(v,v') = (v_i - v'_i)^2$ and maintaining that $\mathbb{T}(x) = \sqrt{x}$ (and $\mathbb{T}^{-1}(x) = x^2$). Here too the regression function learns to associate an importance weight to each deviation in an individual dimension in the search-space.

3. **Weighted Mahalanobis distance** does not preserve the dimensionality of the search-space representation (so that $n = k^2$ where k is the dimension of the search-space). This feature mapping is defined as $\phi_i(v,v') = (v_j - v'_j)(v_l - v'_l)$ where $\mathbb{T}(x) = \sqrt{x}$. If the n elements of $\mathbf{w}$ are reorganized as a matrix A, a linear regression over the defined mapping ϕ can express distances of general quadratic form, $\mathbb{T}(\mathbf{w}\cdot\phi(v,v')) = \sqrt{\sum_{j,l} A_{j,l}(v_j - v'_j)(v_l - v'_l)}$. By incorporating an additional projection step on $\mathbf{w}$, A could be maintained a positive semi-definite (PSD) matrix, which enables importance weights to be assigned to linear combinations of the original search-space rather than to each dimension individually (Shalev-Shwartz *et al.*, 2004). Thus, if the matrix A resulting from reorganizing the elements of $\mathbf{w}$ is PSD, A could be decomposed into $A = B'B$ and $\sqrt{(v-v')A(v-v')} = \|Bv - Bv'\|$. This means that the distance learned by $\mathbf{w}$ is equivalent to measuring Euclidean

distance between, v and v' after both vectors had undergone the linear transformation B. Thus, if on a certain map traffic flows three times faster in the NE-SW axes, the optimal weights must be set to, $\mathbf{w} = (1, \ -\frac{1}{2}, \ -\frac{1}{2}, \ 1)$, and $B = \begin{pmatrix} -\frac{1}{2} & -\frac{1}{2} \\ -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2} \end{pmatrix}$. This is knowledge the first two representations could not acquire.

It is worth while mentioning that the algorithm presented in Figure 2 can be further enriched by incorporating Mercer kernels. Note that the vector $\mathbf{w}$ can be represented as a sum of vectors of the form $\phi(v_i, g_i)$ where $i < t$. We can therefore replace the inner-products in this sum with a general Mercer kernel operator, $K(\phi(v_i, g_i), \phi(v_j, g_j))$.

5 Analysis

We denote by $l_{\mathbf{u}} = l\big(\mathbf{u}; (\phi(t), y_t)\big)$ the loss of a fixed predictor $\mathbf{u} \in \mathbb{R}^n$ to which we are comparing our performance. Our analysis focuses on the realizable case, thus assuming that there exists a vector $\mathbf{u}$ such that $l_{\mathbf{u}} = 0$ for all t. We start with a lemma that provides a bound on the cumulative squared loss of the maximally deviating nodes as a function of $\mathbf{u}$. This lemma is a simple adaptation of Theorem 2 from (Crammer *et al.*, 2006) and is provided in the Appendix for completeness. Next, we follow (Shimbo and Ishida, 2003) and provide the ϵ *additive admissible* lemma, stating that if a heuristic function h never overestimates d by more than a constant value r, then the path returned by A^* using h is guaranteed to be not longer than $d(s, g) + r$. Using these two lemmas we prove that when $\mathbb{T}(x) = x$ and for a sufficiently large T the average deviation of the returned heuristic paths from the optimal ones goes to ϵ. When ϵ goes to zero we obtain convergence to the optimal paths.

Lemma 1 *Let $(\phi(1), y_1), \ldots, (\phi(T), y_T)$ be a sequence of examples where $\phi(t) \in \mathbb{R}^n$, $y_t \in \mathbb{R}$ and $\|\phi(t)\| \leq R$ for all t. Assume that there exists a vector $\mathbf{u}$ such that $l_{\mathbf{u}} = 0$ for all t. Then, the cumulative squared loss on this sequence of examples is bounded by,*

$$\sum_{t=1}^{T} l_{\mathbf{w}_t}^2 \ \leq \ \|\mathbf{u}\|^2 R^2 \ .$$

Lemma 2 *If a heuristic function h never overestimates the true distance d by more than a constant value r, then the path p returned by an A^* search using h is guaranteed to be not greater than $d(s, g) + r$,*

$$\sum_{(v,v') \in p} c(v, v') - d(s, g) \leq r$$

Proof Let $|p| = \sum_{(v,v') \in p} c(v, v')$ be the length of path p. If using the heuristic h, A^* finds a strictly suboptimal path p, then all nodes with an f value less than $|p|$ are expanded while at least some of the nodes within the optimal path are not (or else an optimal path would have been found). Assume for the purpose of contradiction, that the length of the optimal path $d(s, g)$, is smaller than $|p| - r$. Thus, all the nodes on the optimal path have an f value smaller than $|p|$ and must have been expanded while using h, contradicting the fact that a strictly suboptimal path was found. ■

Theorem 1 *When $\mathbb{T}(x) = \mathbb{T}^{-1}(x) = x$, if the conditions of Lemma 1 hold and $T \to \infty$ then the value ϵ bounds the average deviation of the returned paths p_t from the true distances,*

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{(v,v') \in p_t} c(v, v') - d(s_t, g_t) \leq \epsilon \ .$$

Proof Lemma 1 provides that $\sum_{t=1}^{T} l_{\mathbf{w}_t}^2 \ \leq \ \|\mathbf{u}\|^2 R^2$. Dividing by T, we obtain that the average squared loss goes to 0 and therefore the average loss itself goes to 0 as well,

$$\frac{1}{T} \sum_{t=1}^{T} ([|y_t - \mathbf{w_t} \cdot \phi(t)| - \epsilon]_+) \to 0 \ . \qquad (2)$$

Using Eq. (2) and the fact that $[|y_t - \mathbf{w_t} \cdot \phi(t)| - \epsilon]_+ \geq |y_t - \mathbf{w_t} \cdot \phi(t)| - \epsilon$, we obtain the following bound,

$$\frac{1}{T} \sum_{t=1}^{T} (|y_t - \mathbf{w_t} \cdot \phi(t)|) \leq \epsilon \ .$$

Therefore, since the deviances in the learning space are bounded by ϵ, so are the deviances in the search space,

$$\frac{1}{T} \sum_{t=1}^{T} (|h_{\mathbf{w_t}}(v_t, g_t) - d(v_t, g_t)|) \leq \epsilon \ . \qquad (3)$$

Let us define the maximal deviance in search space at round t as, $r_t = |h_{\mathbf{w_t}}(v_t, g_t) - d(v_t, g_t)|$. Using this definition we now average Lemma 2 over all the T rounds, and obtain that,

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{(v,v') \in p_t} c(v, v') - d(s_t, g_t) \leq \frac{1}{T} \sum_{t=1}^{T} r_t \ .$$

From Eq. (3) we know that $\frac{1}{T} \sum_{t=1}^{T} r_t \leq \epsilon$, and therefore,

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{(v,v') \in p_t} c(v, v') - d(s_t, g_t) \leq \epsilon \ .$$

■

If $\mathbf{u}$ can attain a loss of 0 with an ϵ that approaches 0, the returned paths will converge to the optimal ones. The convergence to ϵ is a function of ratio between $\|\mathbf{u}\|^2 R^2$ and

T. Intuitively, $\|\mathbf{u}\|^2 R^2$ indicates the necessary model complexity for correctly characterizing the examples in the online search stream. Thus, although using a sufficiently high dimensional feature mapping ϕ might make a small ϵ feasible, this procedure will typically increase the complexity term $\|\mathbf{u}\|^2 R^2$ by swelling the radius of the training examples R.

6 Experiments

Our experiments aim at examining whether the Online Learning to Search algorithm can return near optimal paths while gradually reducing the number of visited nodes. Experiment 1 focuses on a route planning task, and is aimed at demonstrating that a learned distance adapted to the specific contingencies of the data can have an advantage over a predefined heuristic. Experiment 2 shows that the Online Learning to Search algorithm can prune down the search process without prior domain knowledge. For this, a naive representation of the TopSpin puzzle is applied and the learning mechanism is provided with a large set of automatically generated abstractions. Experiment 3, shows that even in a well studied domain, such as the 8-puzzle, where certain features are known to be effective, the Online Learning to Search algorithm can nevertheless improve performance.

It should be noted that the feedback signal provided in all the reported experiments was derived at the end of each search by running a search process which started at the goal state and continued until exact distances to all of the nodes visited during the current search were evaluated.

6.1 Route Planning

Our first experiment focused on a route planning task where nodes were 231 cities along the East coast of the United States. Graph edges were defined by road distances. Longitude and latitude coordinates of the cities were provided as the source of heuristic information. The selected representation was weighted Euclidean distances. The online sequence of search tasks included 100 trials, each of which was composed of a randomly selected starting city s_t and goal city g_t.

Three heuristics were compared: Dijkstra's search algorithm ($\mathbf{w_t} = (0,0)$), Euclidean distance ($\mathbf{w_t} = (1,1)$) and our Online Learning to Search mechanism. Table 2 displays the average path length and the average number of nodes visited by these three alternatives. It could be seen that the average deviation of the Online Learning to Search algorithm from the optimal path is 1 mile. However, the percentage of visited nodes compared to Dijkstra's algorithm (47%) and to the Aerial distance heuristic (60%) might justify this sub-optimality. Next, we tested the weighted Mahalanobis distance. Using this representation a further reduction in the number of visited nodes was observed (Table 1: full matrix Online Learning to Search). The gradual reduction in the average number of visited nodes is depicted in Figure 3. When averaging $\mathbf{w_t}$ over the last 20 rounds we receive the vector $\mathbf{w} = (16.22, -1.11, -1.11, 10.71)$. Thus the learned metric acquired the fact that traveling along the coast is shorter (in road distance) than traveling in the orthogonal direction.

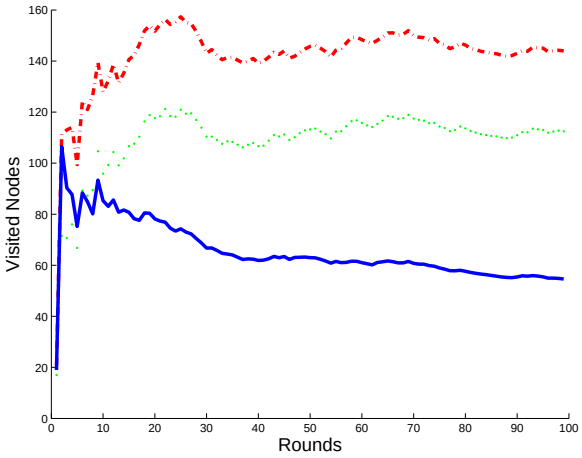

Figure 3: Number of visited nodes averaged over all previous online learning rounds: Dijkstra (dashed), Aerial distance (doted) and Online Learning to Search method using a full matrix (solid).

Table 2: East coast map: path lengths and search extent.

Algorithm	length	# visited
Dijkstra	826	143
Admissible Aerial distance	826	113
Online Learning to Search	827	68
(full matrix)	827	56

6.2 TopSpin

This experiment, focused on a simplified version of the TopSpin puzzle (see www.passionforpuzzles.com/virtualcube/topspin). The naive representation of assigning 1 as the anchor and enumerating clockwise was selected. In this experiment 128 elementary domain abstraction defined ϕ. Each feature counted within an arbitrary set of dimensions, how many mismatches were present between the current state and the desired goal state (ignoring all other dimensions). For example, ϕ_{80} counted mismatches in dimensions 3, 4 and 5. In this experiment, Dijkstra's search algorithm was compared to the traditional Maximal Value composition mechanism and to our Online Learning to Search mechanism. The task of our algorithm was to discover during

100 online rounds, what combination of the candidate features best contributes to the heuristic search.

Table 3 displays the fact that although the features were an automatically generated set of domain abstractions, both heuristic mechanisms were able to prune down the number of visited nodes, while returning the shortest paths. However, the average number of nodes visited by our learned heuristic (88), is significantly smaller than the number of nodes visited by the heuristic search guided by the Maximum Value mechanism (201). This result indicates that different domain abstractions capture different aspects of the distance underlying the search space and thus weighting the individual domain abstractions is preferential to simply selecting the highest scoring feature [2].

Table 3: TopSpin: path lengths and search extent.

Algorithm	length	# visited
Dijkstra	3.4	281
Maximum Value	3.4	201
Online Learning to Search	3.4	88

6.3 8-puzzle

Our last experiment returns to the well studied 8-puzzle, where the (non-admissible) Nilsson sequence score is known to be highly effective in pruning down the search space. This score is defined over two features, $h(v,g) = P(v,g) + 3S(v,g)$. $P(v,g)$ is the Manhattan distance of each tile in v from its proper position in g and the feature $S(v,g)$ is a sequence score obtained by checking around the non-central squares in turn, allotting 2 for every tile not followed by its proper successor and 0 for every other tile (except that a piece in the center scores 1). Thus, in this case the representation is well known and well studied yet the question remains whether the weights assigned for each feature are indeed optimal. The 18 dimensional representation included: 9 Manhattan distance features + 8 binary Nilsson sequence features describing whether each of the peripheral tiles follows the appropriate predecessor + 1 binary feature describing whether the central tile is in place. Here too, 100 rounds of online search tasks were presented. As can been seen in Table 4, the Online Learning to Search is capable of outperforming Nilsson's Heuristic while maintaining admissibility.

[2] Although, it was assumed that the rich features will get the highest weights, these weights were consistently assigned to features of intermediate abstraction (e.g. counting mismatches in 4 elements). This demonstrates that the automated learning process is often free of misleading biases the human designer might possess.

Table 4: 8-Puzzle: path lengths and search extent.

Algorithm	length	# visited
Dijkstra	4.22	148.50
Nilsson's Sequence	4.28	45.90
Online Learning to Search	4.22	15.77

7 Discussion and future extensions

We described a method termed Online Learning to Search, which utilizes state-of-the-art machine learning mechanisms for acquiring a heuristic evaluation function. We relied on the notion of ϵ-admissibility, to prove that when the regression learning task is realizable with a small ϵ then the average divergence from the optimal paths can go to zero. The nature of the regression task ensures that the learned heuristics are highly dominant, in the sense that they effectively prune down the search process. It is important to note that batch regression methods (e.g. Support Vector Regression) can be applied to our setting as well. However, providing formal guarantees using these alternative models is a challenging task. Specifically, it is difficult to see how the i.i.d assumption, which is the cornerstone of statistical inference in the batch setting, might hold in our case (where heuristics must be learned from data with many dependencies).

The proposed online learning mechanism can be extended in several ways. First, the online setting could be applied during a *single search task* by training with distances to already approached nodes, $k(s,v)$. The challenge in the single search setting emerges from learning when only approximate feedback is available. Second, our online mechanism is suitable for tackling scenarios where the optimal weights might be in a continuous state of drift (e.g. accommodating dynamic traffic changes during the day). It is important to emphasize that once the learned heuristic accurately approximates the true distance it is no longer modified and therefore does not require additional feedback (at least until the edge values have been changed). In practice the feedback can actually be given in a rate that is sufficient to follow drifts in the graph characteristics.

The initial step in automating the process of heuristic design followed the observation that formal abstractions of the search space can provide simplified heuristic functions. This paper addresses the question of whether the task of acquiring the appropriate composite heuristic for a certain search space, can be automated as well. We believe that the proposed online algorithm is an initial step in utilizing machine learning for the fundamental challenges posed by artificial intelligence.

Appendix: proof of Lemma 1

Let $(\phi(1), y_1), \ldots, (\phi(T), y_T)$ be an arbitrary sequence of examples, where $\phi(t) \in \mathbb{R}^n$ and $y_t \in \mathbb{R}$ for all $t \leq T$. Define Δ_t to be $\|\mathbf{w_t} - \mathbf{u}\|^2 - \|\mathbf{w_{t+1}} - \mathbf{u}\|^2$. First note that $\sum_t \Delta_t$ is a telescopic sum which collapses to,

$$\begin{aligned}\sum_{t=1}^{T} \Delta_t &= \sum_{t=1}^{T} \left(\|\mathbf{w_t} - \mathbf{u}\|^2 - \|\mathbf{w_{t+1}} - \mathbf{u}\|^2\right) \\ &= \|\mathbf{w}_1 - \mathbf{u}\|^2 - \|\mathbf{w}_{T+1} - \mathbf{u}\|^2.\end{aligned}$$

Using the facts that $\mathbf{w}_1$ is defined to be the zero vector and that $\|\mathbf{w}_{T+1} - \mathbf{u}\|^2$ is non-negative, we can upper bound the right-hand side of the above by $\|\mathbf{u}\|^2$ and conclude that,

$$\sum_{t=1}^{T} \Delta_t \;\leq\; \|\mathbf{u}\|^2 \;. \tag{4}$$

We focus our attention on bounding Δ_t from below on those rounds where $\Delta_t \neq 0$. Using the recursive definition of $\mathbf{w_{t+1}}$, we rewrite Δ_t as,

$$\begin{aligned}\Delta_t &= \|\mathbf{w_t} - \mathbf{u}\|^2 - \|\mathbf{w_t} - \mathbf{u} + \mathrm{sgn}(\delta_t(\mathbf{w_t}))\tau_t\phi(t)\|^2 \\ &= -\mathrm{sgn}(\delta_t(\mathbf{w_t}))2\tau_t(\mathbf{w_t} - \mathbf{u}) \cdot \phi(t) \;-\; \tau_t^2\|\phi(t)\|^2\end{aligned}$$

We now add and subtract the term $\mathrm{sgn}(\delta_t(\mathbf{w_t}))2\tau_t y_t$ from the right-hand side above to get the bound,

$$\begin{aligned}\Delta_t \;\geq\; &+\mathrm{sgn}(\delta_t(\mathbf{w_t}))2\tau_t(\delta_t(\mathbf{w_t})) \\ &-\mathrm{sgn}(\delta_t(\mathbf{w_t}))2\tau_t(\delta_t(\mathbf{u})) \\ &-\tau_t^2\|\phi(t)\|^2 \;.\end{aligned} \tag{5}$$

Since $\mathrm{sgn}(\delta_t(\mathbf{w_t}))\delta_t(\mathbf{w_t}) = |\delta_t(\mathbf{w_t})|$ and since we only need to consider the case where $\Delta_t \neq 0$, then $l_{\mathbf{w}_t} = |\delta_t(\mathbf{w_t})| - \epsilon$ and we can rewrite the bound in Eq. (5) as,

$$\Delta_t \;\geq\; 2\tau_t(l_{\mathbf{w}_t}+\epsilon) - \mathrm{sgn}(\delta_t(\mathbf{w_t}))2\tau_t(\delta_t(\mathbf{u})) - \tau_t^2\|\phi(t)\|^2.$$

We also know that $-\mathrm{sgn}(\delta_t(\mathbf{w_t}))\delta_t(\mathbf{u}) \;\geq\; -|\delta_t(\mathbf{u})|$ and that $-|\delta_t(\mathbf{u})| \;\geq\; -(l_{\mathbf{u}} + \epsilon)$. This enables us to further bound,

$$\begin{aligned}\Delta_t &\geq 2\tau_t(l_{\mathbf{w}_t} + \epsilon) \;-\; 2\tau_t(l_{\mathbf{u}} + \epsilon) \;-\; \tau_t^2\|\phi(t)\|^2 \\ &= \tau_t(2l_{\mathbf{w}_t} - \tau_t\|\phi(t)\|^2 - 2l_{\mathbf{u}}) \;.\end{aligned}$$

Summing the above over all t and comparing to the upper bound in Eq. (4) proves that for any $\mathbf{u} \in \mathbb{R}^n$,

$$\sum_{t=1}^{T} \tau_t \left(2l_{\mathbf{w}_t} - \tau_t\|\phi(t)\|^2 - 2l_{\mathbf{u}}\right) \;\leq\; \|\mathbf{u}\|^2 \;. \tag{6}$$

Using the assumption that the sequence is realizable by the model (there exists a $\mathbf{u}$ for which $l_{\mathbf{u}} = 0$ for all t) and plugging the definition of τ_t into the left-hand side of the above gives,

$$\sum_{t=1}^{T} \frac{l_{\mathbf{w}_t}^2}{\|\phi(t)\|^2} \;\leq\; \|\mathbf{u}\|^2 \;.$$

Now using the fact that $\|\phi(t)\|^2 \leq R^2$ for all t, we get,

$$\sum_{t=1}^{T} l_{\mathbf{w}_t}^2/R^2 \;\leq\; \|\mathbf{u}\|^2 \;.$$

■

References

J. Boyan and A. Moore. Learning evaluation functions to improve optimization by local search. *JMLR*, 2000.

K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *JMLR*, 2003.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *JMLR*, 2006.

J. C. Culberson and J. Schaeffer. Searching with pattern databases. *Advances in Artificial Intelligence (ed. Gordon McCalla), Springer-Verlag, New York*, 1996.

S. Edelkamp and J. Eckerle. New strategies in learning real time heuristic search. *On-line Search: Papers from AAAI Workshop, Providence, RI, AAAI Press*, 1997.

P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 1968.

R.C. Holte and I. Hernadvolgyi. Steps towards the automatic creation of search heuristics. *Technical report TR04-02, Comp. Sci. Depart., Univ. of Alberta*, 2004.

S. Koenig, M. Likhachev, Y. Liu, and D. Furcy. Incremental heuristic search in ai. *AI Mag. 25-2*, 2004.

R. Korf. Real-time heuristic search. *Artificial Intelligence, Vol. 42, No. 2–3, pp. 189– 211*, 1990.

M. Minsky. Steps toward artificial intelligence. *Computers and thought*, 1963.

I. Omer and M. Werman. The bottleneck geodesic: Computing pixel affinity. *CVPR*, 2006.

J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984.

S. J. Russell and P. Norvig. *AI: A Modern Approach*. Prentice Hall, 2nd edition, 2003.

F. Schulz, D. Wagner, and K. Weihe. Dijkstra's algorithm on-line: An empirical case study from public railroad transport. *Lecture Notes in Computer Science. J.S. Vitter, C.D. Zaroliagis (Eds.) p. 110*, 1999.

S. Shalev-Shwartz, Y. Singer, and A. Ng. Online and batch learning of pseudo-metrics. *ICML*, 2004.

M. Shimbo and T. Ishida. Controlling the learning process of realtime heuristic search. *AI 146-1*, 2003.