
Online learning of search heuristics

Michael Fink

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

Abstract

This paper describes a method for efficiently finding a short path between two nodes in a graph, by minimizing the number of nodes visited during the search. The main contribution of this paper is an automated mechanism for learning a heuristic function that can guide the search in the right direction. Our heuristic function is parameterized by a vector of importance weights over a set of features, which are typically simple abstractions of the search space. The importance weights are trained in an online manner using nodes to which the true distance has already been revealed during previous search stages. Our experiments demonstrate that the proposed method typically finds the optimal path while significantly reducing the search complexity. A theoretical analysis describes conditions under which finding the shortest path can be guaranteed.

1 Introduction

Searching a shortest path between two nodes in a graph is a classical problem in computer science, emerging in many domains ranging from train scheduling [12] to computer graphics [10]. The well known Dijkstra's shortest path search algorithm remains one of the top 10 cited computer science papers many years after it was first published. However, the time complexity of Dijkstra's algorithm, $O(|V|^2)$ is intolerable if a quick answer is required while searching a very large graph. Two complementary lines of research have responded to the challenge of efficiently finding short paths in large graphs.

The first line of research focuses on improving the priority queue data structure used by Dijkstra's algorithm, while attempting to make use of different graph properties. For example, it has been demonstrated that if the graph is sparse using a binary heap, the algorithm requires

$O((|E| + |V|)\log|V|)$ time, and using the Fibonacci heap improves this to $O(|E| + |V|\log|V|)$.

The second line of research introduced a heuristic factor into the search process. A prototypical representative of this line of research, is the A^* algorithm (originally suggested by [5]). The assumption guiding A^* , is that in addition to the search problem variables (graph, starting node and target node), the algorithm receives as input a heuristic function which is used to evaluate the remaining distance from each visited node to the target node. Adding this heuristic score to the nodes stored in Dijkstra's priority queue can guide the search towards the target and thus avoid visiting a substantial part of the graph. The A^* algorithm can guarantee that the shortest path is found when the heuristic function h never overestimates the true distance from a certain node to the target. Since $h = 0$ is always admissible, the admissibility guarantee seems to be meaningless without the notion heuristic dominance. Heuristic dominance provides that an admissible heuristic h which consistently provides higher estimations than the heuristic function h' , will never visit during the search more nodes than h' (see [11] and references within).

The paper by [15] associated tasks that require intelligence with heuristic search. However, the intelligence applied in solving these tasks should in fact be attributed to the designer of the successful heuristic. This paper addresses the question of whether the task of designing an appropriate heuristics for a certain search space, might be automated. As was stated above, Dijkstra's algorithm is still widely used despite the existence of A^* , which seems to indicate that the challenge of heuristic design is a significant barrier in the application of heuristic search algorithms. Moreover, in certain settings 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, when vehicle s must communicate with a distant vehicle g , the goal is to transmit the conversation 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-optimal¹.

The initial step in automating the process of heuristic design followed the observation that abstractions of the search space can provide simplified evaluation functions [9]. This theme, first suggested in the 60-s has been developed throughout the years with increasing sophistication [6]. Thus, elementary heuristics derived from by domain abstraction can be automatically composed into complex admissible heuristics (e.g. by taking the maximum value of the elementary heuristics). This paper builds upon the notion of domain abstraction and suggests a family of algorithms aimed at automating the heuristic composition process, utilizing state-of-the-art online learning mechanisms.

Our setting assumes that a sequence of related search tasks must be performed using an A^* mechanism and that at the end of each search task the currently held composite heuristic receives feedback indicating which of its estimates have maximally deviated from the true distances. Using this feedback the parameters of the heuristic are updated so it can better capture the characteristics of the distance underlying the specific sequence of observed graphs. Our setting is related to other repeated graph search setting [8, 3, 4, 7]. However, while these methods rely on memorization for transferring heuristic knowledge, our method relies on the machine learning notion of generalization.

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 target 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 target node and traverse the graph 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 $(G_1, s_1, g_1) \dots (G_T, s_T, g_T)$ 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 reply by producing the edges on the shortest path p_t for reaching g_t from s_t .

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*,

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

It is assumed that a function $\phi : V \times V \rightarrow \mathbb{R}^n$ is provided, which receives a pair of nodes (v, v') from the search space and returns an n dimensional feature mapping in the learning-space. Each feature ϕ_i is a search-space abstraction, typically a simplification generated by ignoring some of the domain constraints. The regression function multiplies the features in 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 $y_v = \mathbb{T}^{-1}(d(v, g))$,

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

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.

¹Throughout the paper we assume that the graph might be dynamically changing but that these changes occur in a longer time constant than that of the search process.

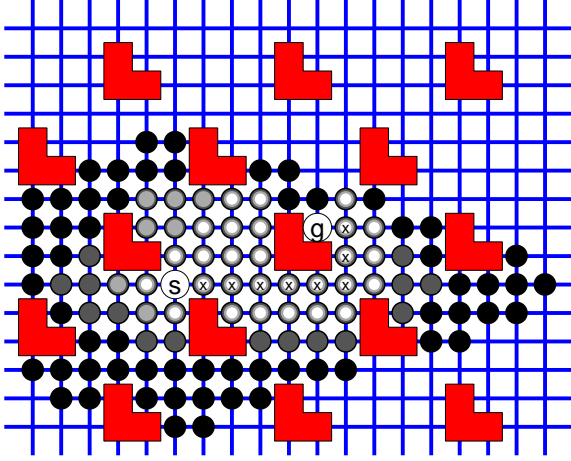


Figure 1: A graph of the road grid in Anytown, USA. The L-shaped structures represent shopping malls. Nodes on the shortest path from s to g are indicated by \times . Notice the increasing heuristic dominance between: Dijkstra (Black), Block Distance (Dark Gray), Optimal Admissible (Light Gray) and Online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

For concreteness let us review an example using the map in Figure 1, depicting the road grid and the shopping malls 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 target 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, $\phi_1(v_x, v'_x) = |v_x - v'_x|$ and $\phi_2(v_y, v'_y) = |v_y - v'_y|$, 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. In Figure 1 the nodes visited by different algorithms are color coded: Black - Dijkstra $\mathbf{w} = (0, 0)$, Dark Gray - Block Distance $\mathbf{w} = (1, 1)$, Light Gray - Optimal Admissible $\mathbf{w} = (1, 3)$ and in White is 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 target 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 target (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 in maintaining that in round t the heuristic function h is:

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

The first obstacle in achieving these goals is that if the T search problems presented during the online stream are unrelated, anything learned during the first $t - 1$ rounds might be irrelevant on round t . It is therefore necessary to characterize the conditions under which a heuristic function learned from distance feedback on certain graphs, can fulfill the two criteria stated above, while searching on a new graph.

The second obstacle in achieving our two stated goals, results from the fact that these goals guide towards opposing trends. By approximating the true distance function $d(v, v')$ the heuristic function $h(v, v')$ is maximally dominant and the number of nodes visited during the search can be minimized. However, approximating $d(v, v')$ jeopardizes the path optimality, since $h(v, v')$ can occasionally overestimate $d(v, v')$ and thus be rendered non-admissible.

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 [2]. 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 h has the capacity to approximate the true distances $d(v, v')$ up to a small constant, ϵ . The more complex mechanisms proposed in [2] have the advantage of relaxing this assumption and will be briefly mentioned later on.

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 ϕ_t) and the target value is $y_t = \mathbb{T}^{-1}(d(v_t, g_t))$. We will address such pairs (ϕ_t, y_t) , as our learning examples. Our algorithm relies on the ϵ -insensitive hinge loss function:

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

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 $|\mathbf{w} \cdot \phi - y|$.

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

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

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 rationale 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,

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \frac{\operatorname{sign}(\delta_t(\mathbf{w}_t)) \tau_t}{\|\phi_t\|} \phi_t.$$

It should be noted that [2] provide modifications of this 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

INPUT: $\phi(v, v')$ feature mapping
 \mathbb{T} learn-space to search-space reversible transformation
 ϵ learning feasibility parameter

INITIALIZE: $\mathbf{w}_1 \leftarrow \mathbf{0}$

For $t = 1, 2, \dots$

define current search 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 $l_{\mathbf{w}_t} \leftarrow [|y_t - \mathbf{w}_t \cdot \phi_t| - \epsilon]_+$

If $l_{\mathbf{w}_t} > 0$

set: $\tau_t = \frac{l_{\mathbf{w}_t}}{\|\phi_t\|^2}$

update: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \operatorname{sign}(y_t - \mathbf{w}_t \cdot \phi_t) \tau_t \phi_t$

Figure 2: The online learning to search algorithm.

where the feedback signal provided at each round includes *all* the true distances rather than just the most deviant one [1]. 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 tradeoff 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 the appropriate reversible non-decreasing transfer functions \mathbb{T} :

1. **Weighted Block distance:** is formally defined as, $\phi_i(v, v') = |v_i - v'_i|$, where ϕ_i , indicates the i th 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 representations.

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 ϕ can express distances of general quadratic form, $\mathbb{T}(\mathbf{w} \cdot \phi(v, v')) = \sqrt{\sum_{j,l} A_{j,l}(v_j - v'_j)(v_l - v'_l)}$. With some constraints on the learning process of \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 [13]. Thus, if the matrix A resulting from reorganizing the elements of \mathbf{w} is PSD, A could be decomposed into $A = B^T B$ and $\sqrt{(v - v')^T A (v - v')} = \|Bv - Bv'\|$. 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 ².

Finally it is worth while mentioning that the algorithm from 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 $l_{\mathbf{u}} = l(\mathbf{u}; (\phi_t, y_t))$ 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 loss bound on the cumulative squared loss of the maximally deviating nodes. This lemma is a simple adaptation of Theorem 2 from [2] and is provided in the Appendix for completeness. Next, we follow [14] 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 $(\phi_1, y_1), \dots, (\phi_T, y_T)$ be a sequence of ex-*

²If on a certain map the optimal weights are, $\mathbf{w} = (1, -\frac{1}{2}, -\frac{1}{2}, 1)$, implying that $B = \begin{pmatrix} -\frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$, then the learned representation found the appropriate heuristic for traffic that flows three times faster in the NE-SW axes. This is knowledge the first two representations could not have acquired.

amples 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 We abbreviate the length of path p as $|p| = \sum_{(v, v') \in p} c(v, v')$. If using the heuristic h , A^* returns a strictly suboptimal path p then all nodes with an f value smaller than $|p|$ have been expanded. In addition some of the nodes within the optimal path have not been expanded (or an optimal path would have been found). Let us 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 . This contradicts the fact that A^* using h found a strictly suboptimal path. ■

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

$$\langle \sum_{(v, v') \in p_t} c(v, v') - d(s_t, g_t) \rangle_t \leq \epsilon .$$

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, $\langle (|y_t - \mathbf{w}_t \cdot \phi_t| - \epsilon)_+^2 \rangle_t \rightarrow 0$. And therefore it holds that the average loss itself goes to 0 as well,

$$\langle (|y_t - \mathbf{w}_t \cdot \phi_t| - \epsilon)_+ \rangle_t \rightarrow 0 . \quad (2)$$

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,

$$\langle (|y_t - \mathbf{w}_t \cdot \phi_t|) \rangle_t \leq \epsilon .$$

Therefore, since the average loss is bounded by ϵ , so is the maximal deviance in the search space,

$$\langle (|h_{\mathbf{w}_t}(v_t, g_t) - d(v_t, g_t)|) \rangle_t \leq \epsilon . \quad (3)$$

Let us define the maximal divergence 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,

$$\langle \sum_{(v,v') \in p_t} c(v, v') - d(s_t, g_t) \rangle_t \leq \langle r_t \rangle_t .$$

Since from Eq. (3) we know that $\langle r_t \rangle_t \leq \epsilon$ we conclude that,

$$\langle \sum_{(v,v') \in p_t} c(v, v') - d(s_t, g_t) \rangle_t \leq \epsilon .$$

■

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 complexity of 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.

6 Experiments

Our experiments were aimed at examining whether the Online Learning to Search algorithm can return near optimal paths while using the feedback signal to gradually reduce the number of visited nodes. Experiment 1 focuses on a path finding 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 abstract representations. Experiment 3, shows that even in a well studied domain, such as the 8-puzzle, where certain abstractions are known to be effective, the Online Learning to Search algorithm can nevertheless, contribute to improving 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 Dijkstra process which started at the goal state and continued until exact distances to all of the nodes visited during the current search were evaluated. Note, that approximate feedback can be obtained by replacing the Dijkstra process with a heuristic search originating at the goal node g and traversing back to start node s . Preliminary experimentation in this setting shows results which are surprisingly similar to those obtained by the exact feedback process.

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 and Canada. 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 a randomly selected and goal city g_t . Performance of three A^* heuristics was 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.

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

In a second variation of this experiment the representation was modified to the weighted Mahalanobis distance. Using this representation a further reduction in the number of visited was observed (Table 1: full matrix Online Learning to Search). The learning process leading to a 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 $\mathbf{w} = (16.22, 10.71, -1.11)$. This result implies that the learned metric acquired the fact that traveling in the North-South axis (along the coast) is on average shorter (in road distance) than traveling in the orthogonal direction.

6.2 TopSpin

Our second experiment, focused on a simplified version of the TopSpin puzzle (see www.passionforpuzzles.com/virtualcube/topspin). The naive representation of using the number 1 as anchor and enumerating clockwise was selected. In this experiment 128 elementary domain abstraction were incorporated into ϕ . Each of these features i counted within an arbitrary set of dimensions how many mismatches were present between the current state and the desired goal state (while

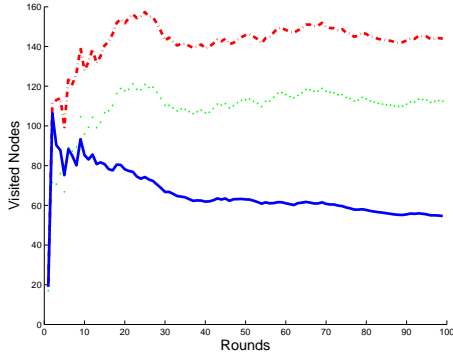


Figure 3: Average cumulative number of visited nodes: Dijkstra (dashed), Aerial distance (dotted) and Online Learning to Search method (solid).

ignoring all other dimensions). For example, ϕ_{100} only counted mismatches in dimensions 3, 4 and 5. Each of these dimensions is an admissible heuristic. The task of the Online Learning to Search mechanism was to discover during the 100 online rounds, which of the large set of candidate features truly contributes to the heuristic search. Table 3 displays the fact that although the abstract features were an arbitrary selection, the Online Learning to Search mechanism, managed to tune onto a set of weights that maintained admissibility, while significantly pruning down the search process.³

Table 3: TopSpin: path lengths and search extent.

Algorithm	length	# visited
Dijkstra	3.4	281
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, $h(v, g) = P(v, g) + 3S(v, g)$. The feature $P(v, g)$ 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

³It was (wrongly) assumed that the features that observe the maximal number of elements will be most informative and get the highest weights. Surprisingly, the highest weights were consistently assigned to features of intermediate abstraction (e.g. counting mismatches in 4 elements). The challenge of interpreting this results remains open, however it is apparent that the automated learning process can occasionally be free of misleading biases the human designer might possess.

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 task were presented. As can be seen in Table 4, the Online Learning to Search is capable of significantly outperforming Nilsson’s Heuristic while maintaining admissibility.

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 setting (where the evaluation functions must be learned and applied to 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. The rounds in this online setting could be derived either by a lookahead procedure or by using the known length, $k(s, v)$, of reaching a visited node v from node s . 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). Finally, we believe that the proposed online algorithm is an initial step towards apply-

ing machine learning techniques to the fundamental challenges of artificial intelligence.

Appendix: proof of Lemma 1

Let $(\phi_1, y_1), \dots, (\phi_T, y_T)$ be an arbitrary sequence of examples, where $\phi_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}$ for all t . 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,

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

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,

$$\sum_{t=1}^T \Delta_t \leq \|\mathbf{u}\|^2. \quad (4)$$

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,

$$\begin{aligned} \Delta_t &= \\ \|\mathbf{w}_t - \mathbf{u}\|^2 - \|\mathbf{w}_t - \mathbf{u} + \text{sign}(\delta_t(\mathbf{w}_t))\tau_t\phi_t\|^2 &= \\ -\text{sign}(\delta_t(\mathbf{w}_t))2\tau_t(\mathbf{w}_t - \mathbf{u}) \cdot \phi_t - \tau_t^2\|\phi_t\|^2 & \end{aligned}$$

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

$$\begin{aligned} \Delta_t &\geq \\ +\text{sign}(\delta_t(\mathbf{w}_t))2\tau_t(\delta_t(\mathbf{w}_t)) & \\ -\text{sign}(\delta_t(\mathbf{w}_t))2\tau_t(\delta_t(\mathbf{u})) & \\ -\tau_t^2\|\phi_t\|^2. & \end{aligned}$$

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

$$\Delta_t \geq 2\tau_t(l_{\mathbf{w}_t} + \epsilon) - \text{sign}(\delta_t(\mathbf{w}_t))2\tau_t(\delta_t(\mathbf{u})) - \tau_t^2\|\phi_t\|^2.$$

We also know that $-\text{sign}(\delta_t(\mathbf{w}_t))\delta_t(\mathbf{u}) \geq -|\delta_t(\mathbf{u})|$ 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 $\mathbf{u} \in \mathbb{R}^n$,

$$\sum_{t=1}^T \tau_t (2l_{\mathbf{w}_t} - \tau_t\|\phi_t\|^2 - 2l_{\mathbf{u}}) \leq \|\mathbf{u}\|^2. \quad (5)$$

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

- [1] K. Crammer and Y. Singer. Ultraconservative online algorithms for multiclass problems. *JMLR*, 2003.
- [2] K. Crammer and O. Dekel and J. Keshet and S. Shalev-Shwartz and Y. Singer. Online Passive-Aggressive Algorithms *Technical Report, Leibniz Center*, 2005.
- [3] J. C. Culberson and J. Schaeffer. Searching with pattern databases. *Advances in Artificial Intelligence (ed. Gordon McCalla), Springer-Verlag, New York*, 1996.
- [4] 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.
- [5] P. E. Hart and 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
- [6] R. C. Holte and I. Hernadvolgyi. Steps Towards the Automatic Creation of Search Heuristics. *technical report TR04-02, Computing Science Department, University of Alberta*, 2004.
- [7] S. Koenig and M. Likhachev, Y. Liu, and D. Furcy. Incremental heuristic search in AI. *AI Mag. 25 (2) pp. 99–112, American Association for Artificial Intelligence, Menlo Park, CA.*, 2004.
- [8] R. Korf Real-Time Heuristic Search. *Artificial Intelligence, Vol. 42, No. 2–3, pp. 189–211*, 1990.
- [9] M. Minsky. Steps toward artificial intelligence. *Computers & thought*, 1963.
- [10] I. Omer and M. Werman. The Bottleneck Geodesic: Computing Pixel Affinity. *CVPR*, 2006.
- [11] S. J. Russell and P. Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd edition, 2003.
- [12] F. Schulz and 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.
- [13] S. Shalev-Shwartz, Y. Singer and A. Ng. Online and Batch Learning of Pseudo-Metrics. *ICML*, 2004.
- [14] M. Shimbo and T. Ishida. Controlling the learning process of real-time heuristic search. *Artif. Intell. 146 (1)*, 2003.
- [15] H. A. Simon. Artificial Intelligence: An Empirical Science. *Artificial Intelligence 77, 95-12*, 1995.