Learning Heuristic Functions Through Approximate Linear Programming

Marek Petrik and **Shlomo Zilberstein**
Department of Computer Science
University of Massachusetts Amherst
Amherst, MA 01003
{petrik,shlomo}@cs.umass.edu

Abstract

Planning problems are often formulated as heuristic search. The choice of the heuristic function plays a significant role in the performance of planning systems, but a good heuristic is not always available. We propose a new approach to learning heuristic functions from previously solved problem instances in a given domain. Our approach is based on approximate linear programming, commonly used in reinforcement learning. We show that our approach can be used effectively to learn admissible heuristic estimates and provide an analysis of the accuracy of the heuristic. When applied to common heuristic search problems, this approach reliably produces good heuristic functions.

Introduction

Heuristic search plays an important role in planning (Bonet and Geffner 2001; Hoffman and Nebel 2001). Numerous planning algorithms have been developed based on A*, branch an bound, and their many derivatives. These algorithms use an *admissible heuristic* function to guide the search process and prune unpromising parts of the search space. Thus, a good heuristic function is crucial for good performance of many planning systems. When a planning problem is formulated as an integer program, an admissible heuristic can be derived using the corresponding linear program without the integrality constraints (Bylander 1997). Yet, many planning domains lack the necessary structure that makes such relaxation methods applicable. As a result, heuristic functions are often constructed manually.

There has been significant progress in recent years with automatic construction of heuristic functions using state abstraction in either deterministic (Holte et al. 1996) or stochastic domains (Beliaeva and Zilberstein 2005). Effective methods have been developed based on hierarchical heuristic search (Holte, Grajkowski, and Tanner 2005) and pattern databases (Culberson and Schaeffer 1996). By applying abstraction to the original state-space, a much smaller problem can be created, the solution of which provides an admissible heuristic estimate for solving the original problem. The key question is how to construct the abstract state space efficiently. Some state aggregation techniques work well in domains with a certain structure, but extensive analysis or enumeration of the original state-space are required in others. Thus creating a good heuristic function automatically in large domains–ones that cannot be fully enumerated–remains an important challenge. We show how this problem can be solved using sample plans. Our approach produces quickly a heuristic function that is admissible and precise with high probability.

In many settings, such as online planing and conformant planning, problem instances must be solved multiple times. The results of previous searches–both the solutions and expanded nodes–offer valuable statistics on the performance of the heuristic function, and can be used to improve it. We propose an approach to construct a heuristic function automatically from search data. The goal is not necessarily to improve on the best known heuristic functions for well-studied domains, which are often based on thorough analysis and deep human insights. Instead, the advantage of our approach is its generality and that it does not impose restricting conditions. Therefore, it is very easy to apply it to new domains to quickly obtain useful admissible heuristics.

We propose a new framework to create heuristic functions based on approximate linear programming (ALP). ALP is often used in reinforcement learning to learn how to act in complex stochastic domains modeled as Markov decision processes (MDPs) (Powell 2007). Solution techniques are typically based on calculating a *value function*, which is the expected total return that can be obtained when acting optimally starting in a given state. The value function is typically maximized. To solve large problems, the value function can be approximated using a small linear subspace, generated by an approximation *basis*. The basis captures a limited set of features of all states. Given that the basis is restricted, the value function needs to be solved only for a small subset of all states. The states can be sampled and only a small portion of them is typically required, depending on the size of the subspace.

Approximate linear programming is beneficial for constructing heuristic functions for several reasons. In ALP, the Markov decision problem is formulated as a linear program in which the variables represent the values of the different states (Powell 2007). To approximate the value function, the variables are restricted to a small-dimensional linear space. This makes the problem much easier to solve. While in some

small problems it is possible to use all the constraints, it is typically sufficient to only use a small subset of them. This is because of properties of linear programs which we discuss below. Furthermore, ALP is guaranteed to produce an upper bound on the exact value function (Trick and Zin 2005). Since in MDPs the goal is to maximize the expected reward, an upper bound provides an admissible heuristic. The restriction to a small-dimensional linear space ensures that the heuristic function is easy to compute.

Approximate linear programming can be seen as a generalization of state-space abstraction and additive pattern databases. Like state space abstraction, the method relies on grouping states, based on their features. It does provide, however, a richer set of representations, as we describe below. Furthermore, it does not require a full enumeration of the abstract search space. Instead, the approach may be based purely on samples of the transitions (de Farias and Roy 2004; Goldfarb and Iyengar 2003; Ben-Tal and Nemirovski 2008). Because the features in ALP may be real numbers, it is possible to use pattern databases as the features. Hence, our approach can be seen as finding a linear combination of pattern databases that preserves the admissibility of the heuristic function.

The paper is organized as follows. First, we formally define the framework. Next, we describe a basic formulation of the approximate linear program for finding an admissible heuristic function. Then, we propose an alternative formulation, which minimizes the maximal error of the heuristic function. Properties of the approximation features that ensure a good approximation are described next. This extends the standard analysis of approximate linear programming and provides a new foundation for automatic creation of good admissible heuristics. Finally, we demonstrate the approach on the 8-puzzle problem–a simple well-known problem that allows us to analyze the quality of the obtained heuristic functions.

Formal Framework

We begin with a formal description of the search problem. To maintain consistency with the standard reinforcement learning literature, we assume that the problem is a maximization problem. Planning problems are often framed as minimization problems, but any minimization problem can be easily transformed to a maximization one by using negative rewards to capture the cost of actions.

While we focus in this paper on deterministic problems, the framework also applies to stochastic domains, when used with LAO* (Hansen and Zilberstein 2001). In fact, it is harder to find admissible heuristics in stochastic domains so our approach may prove particularly useful (Beliaeva and Zilberstein 2005). The search problem is defined as follows:

Definition 1. A search problem is a tuple $(\mathcal{S}, \mathcal{A}, t, r)$, where $\mathcal{S}$ is a set of states with a single initial state σ and a single goal state τ. $\mathcal{A}$ is a set of actions available in each state, and the function $t(s, a) = s'$ specifies the successor state s' of some state $s \neq \tau$ when action a is taken. The function $r(s, a) \in \mathbb{R}$ specifies the corresponding reward, which is a negative cost.

The objective is to find a terminating path for a given state s with the maximal *discounted cumulative* return. The path is a sequence of the form $(\{a_i \in \mathcal{A}, s_i \in \mathcal{S}\})_{i=1}^{n}$, such that $s_n = \tau$ and $t(s_i, a_i) = s_{i+1}$ for $i < n$. The *value* of such a sequence, starting in some state s_1, is:

$$v(s_1) = \sum_{i=1}^{n-1} \gamma^i r(s_i, a_i).$$

The optimal value, denoted as $v^*(s_1)$, is the maximal possible value over any goal-terminated sequence starting in the given state. The discount factor is considered only for the sake of generality. Unlike some traditional MDP solution techniques, our approach applies also to problems without discounting (i.e., when $\gamma = 1$). We assume that there is always a path from σ to any state s and that there is a path from any state s to the goal τ. We also assume that there are no cycles with positive cumulative rewards. Otherwise, it is possible to construct a solution with an infinite return when $\gamma = 1$. Finally, we assume that the goal is reachable from every state.

A *heuristic function* $h : \mathcal{S} \to \mathbb{R}$ is a function that assigns a real value to each state, which is an estimate of the value of that state. Since we consider maximization problems, a heuristic function is admissible when $h(s) \geq v(s)$ for all $s \in \mathcal{S}$. In general, a lower admissible heuristic function is preferable because it tends to be a more accurate estimate of v. That is, the heuristic function is an upper bound on the value of each state. We also consider a lower bound on the value of each state, denoted as $\theta : \mathcal{S} \to \mathbb{R}$, such that $\theta(s) \leq v(s)$ for all $s \in \mathcal{S}$. The utility of such a function is to asses the bound on the approximation error of the heuristic function.

The heuristic function is learned from samples of previously solved problem instances. These are sequences of state-action pairs that connect a state with the goal. We assume that some of them are *arbitrary* goal-terminated paths:

$$(\{s_i^j, a_i^j\})_i^{n_j} \quad j \in B, s_{n_j} = \tau,$$

and others are *optimal* goal-terminated paths:

$$(\{s_i^j, a_i^j\})_i^{n_j} \quad j \in A, s_{n_j} = \tau.$$

The sequences in both sets A and B are used to determine the lower bounds on the heuristic function. That is, they ensure that the heuristic function is admissible. The sequences in set A are used to determine the lower bounds on v^* and thus how far the heuristic function is from the true value of each state. We denote the set of all sequences $C = A \cup B$. Notice that in the following, s_i^j denotes an i-th state of the sequence j, while s_i is a specific state from $\mathcal{S}$, unrelated to s_i^j. A similar notation is used for actions.

A necessary component for constructing a heuristic function is a set of state features. The heuristic function is obtained as a linear combination of the features, and thus the set of feasible heuristic functions is a linear space. Because the features abstract the state space, they enable generalization from an incomplete set of samples. The basis of this space is composed of the columns of a matrix denoted as

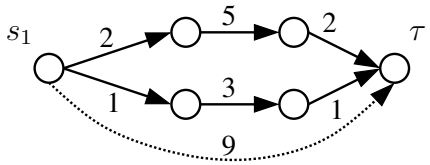

Figure 1: Formulations ensuring admissibility.

M, with each row corresponding to a single state. That is, each row of M defines the features for the corresponding state. The features are not limited to $\{0, 1\}$, but may have arbitrary real values. The heuristic function is expressed as:

$$h = Mx,$$

for some vector x, which represents the weight on the features. We use the vector and function notations interchangeably, depending on the circumstances. In the vector notation, we assume an arbitrary but fixed ordering of the states and the actions. A specific instance of such approximation is *abstraction*, or aggregation. In abstraction, the set of all states $\mathcal{S}$ is partitioned into $\mathcal{S}_1 \dots \mathcal{S}_m$. The approximation basis is then constructed as: $M(i, j) = 1 \Leftrightarrow s_i \in \mathcal{S}_j$, where s_i corresponds to row i. Choosing the basis to be the identity matrix allows an arbitrary heuristic function to be expressed.

Besides reducing the need for sampling, a limited basis allows for heuristic functions to depend on features that are calculated easily. Without any restriction, the heuristic function in problems with a complex state space could be as hard to compute as the actual solution.

Admissible Heuristic Functions

In this section we propose two linear formulations that ensure the admissibility of a heuristic function. The feasible set is represented by a set of linear inequalities.

The two basic formulations are depicted in Figure 1. The first formulation, is to simply bound the heuristic value by the value received in the sampled states, and is represented by the dotted line in the figure. Formally, this is stated as:

$$h(s_i^j) \geq \sum_{k=i}^{n_j} \gamma^{k-i} r(s_k^j, a_k^j) \quad \forall j \in C, \forall i = 1 \dots n_j \quad (1)$$

Clearly, this formulation ensures the admissibility of the heuristic function. Notice that this will possibly mean multiple inequalities for each state, but only the dominating ones need to be retained. Thus let v_i denote the highest right-hand side for state s_i. The function must be restricted to the vector subspace spanned by columns of M. For notational convenience, we formulate the problem in a way that is independent of samples. Let h and v be column vectors with each row corresponding to a state. Then the inequality may be written as:

$$h = Mx \geq v,$$

treating h as a vector. In general, only some of the inequalities are provided based on the available samples; with all samples $v = v^*$. To simplify the notation, we denote this feasible set as H_1, and thus $h \in H_1$.

The second formulation is based on approximate linear programming, and is represented by the solid lines in Figure 1. In this case, the sample paths do not need to be terminated by the goal node. However, the heuristic function is actually required to be *consistent*, which is a stronger condition than admissibility. That is, for each observed sequence of two states, the difference between their heuristic values must be greater than the reward received. Formally,

$$\begin{aligned} h(s_i^j) &\geq \gamma h(s_{i+1}^j) + r(s_i^j, a_i^j) \\ &\forall j \in C, \forall i = 1 \dots n_{j-1} \\ h(\tau) &\geq 0 \end{aligned} \quad (2)$$

In this case, we can define an action-transition matrix T_a for action a. The matrix captures whether it is possible to move from the state defined by the row to the state defined by the column.

$$T_a(i, j) = 1 \Leftrightarrow t(s_i, a) = s_j.$$

A transition matrix T for all actions can then be created by vertically appending these matrices as follows:

$$T = [T_{a_1}; T_{a_2} \dots].$$

Similarly, we define a vector r_a of all the rewards for action a, such that $r_a(i) = r(s_i, a)$. The vector r of all the rewards for all the actions can then be created by appending the vectors: $r = [r_{a_1}; r_{a_2} \dots]$. The constraints on the heuristic function in matrix form become:

$$h \geq \gamma T_a h + r_a \quad \forall a \in \mathcal{A},$$

together with the constraint $h(\tau) \geq 0$. To include the basis M to which the heuristic function is constrained, the problem is formulated as:

$$\begin{aligned} (I - \gamma T_a) M x &\geq r_a \quad \forall a \in \mathcal{A} \\ h(\tau) &\geq 0, \end{aligned}$$

where I is the identity matrix. To simplify the notation, we denote the feasible set as H_2, and thus $h \in H_2$.

The formulation in Eq. (2) ensures that the resulting heuristic function will be admissible.

Proposition 1. *Given a complete set of samples, the heuristic function $h \in H_2$ is admissible. That is, for all $s \in \mathcal{S}$, $h(s) \geq v(s)$.*

Proof. By induction on the length of the path from state s to the goal with maximal value. The base case follows from the definition. For the inductive case, let the optimal path to the goal from state s take action a, breaking ties arbitrarily. Let then $s_1 = t(s, a)$. From the inductive hypothesis and sub-path optimality, we have that $h(s_1) \geq v(s_1)$. Then:

$$h(s) \geq \gamma h(s_1) + r(s, a) \geq \gamma v(s_1) + r(s, a) = v(s).$$

Therefore, for a finite state space, the function h is an admissible heuristic function. □

In addition to admissibility, given incomplete samples, the heuristic function obtained from Eq. (2) is guaranteed not to be lower than the lowest heuristic value feasible in Eq. (1), as the following proposition states.

Proposition 2. *Let C be a set of samples that does not necessarily cover all states. If h is infeasible in Eq. (1), then h is also infeasible in Eq. (2).*

The proof of this proposition is simple and relies on the fact that if an inequality is added for every segment of a path that connects it to the goal, then the value in this state cannot be less than the sum of the transition rewards.

Proposition 2 shows that given a fixed set of samples, Eq. (2) guarantees admissibility whenever Eq. (1) does. However, as we show below, it may also lead to a greater approximation error. We therefore analyze a hybrid formulation, weighted by a constant α:

$$\begin{aligned} &\forall j \in C, \forall i = 1 \dots n_{j-1} : \\ &\quad h(s_i^j) \geq \ \alpha\gamma h(s_{i+1}^j) + \alpha r(s_i^j, a_i^j) + (1-\alpha)v(s_i^j) \end{aligned} \tag{3}$$

Here $v(s_i^j)$ is the value of state s_i^j in sequence j. When it is not available, an arbitrary lower bound may be used. For $\alpha = 0$, this formulation is equivalent to Eq. (1), and for $\alpha = 1$, the formulation is equivalent to Eq. (2). We denote the feasible set as H_3, and thus $h \in H_3$. The key property of this formulation is stated in the following lemma, which is used later in the paper to establish approximation bounds, and is straightforward to prove.

Lemma 1. *The optimal value function v^* is a feasible solution of Eq. (3) for an arbitrary α.*

Given the above, we are ready to formulate the linear program for an admissible heuristic function. In general, it is beneficial to make the heuristic function as close as possible to the optimal value, while preserving its admissibility. That implies minimizing the heuristic function, while ensuring admissibility:

$$\begin{aligned} &\text{minimize} \quad c^T h \\ &\text{subject to} \quad h \in H_3 \end{aligned} \tag{4}$$

In this program, c is an arbitrary non-negative vector that sums to one. It represents a distribution over the states. The formulation corresponds exactly to approximate linear programming when $\alpha = 1$. But to use this program in practice, we need to address two fundamental questions:

1. Minimizing the weighted sum of the errors is somewhat arbitrary (de Farias 2002); and
2. The existence of a finite admissible heuristic is not guaranteed. That is, Eq. (4) may not have a feasible solution.

We address these issues below and discuss the performance of these methods under partial availability of samples.

Lower Bounds on the Value Function

In this section we show that in some settings it is possible to obtaining a tight lower bound on the value of each state. This is important because it allows us to evaluate the difference between the heuristic value and the true value of each state. The lower bounds on the values of some selected states are obtained from the optimal solutions.

The formulation we consider is similar to Eq. (1).

$$\theta(s_i^j) \leq \sum_{k=i}^{n_j} \gamma^{k-i} r(s_k^j, a_k^j) \quad \forall j \in A, \forall i = 1 \dots n_j \tag{5}$$

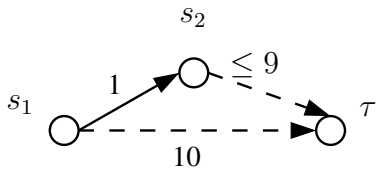

Figure 2: Lower bound formulations, where the dotted lines represent paths of arbitrary length.

That is, the bounds are on the values of states that were solved optimally and any nodes that are on the path connecting the start state with the goal state. These bounds can also be written in matrix notation, as in Eq. (1):

$$\theta = My \geq v^*.$$

We denote this feasible set G_1, and thus $\theta \in G_1$. Additional bounds may be introduced as well. Given an admissible heuristic function, bounds can be deduced for any state that is expanded, even when it is not on an optimal path. While these bounds may not be tight in many cases, they will only increase the probability that the function θ is a lower bound. Notice that these constraints are sampled in the same manner as the constraints that ensure feasibility.

Proposition 3. *When the set of samples is complete and θ satisfies Eq. (5), then*

$$\theta(s) \leq v^*(s) \quad \forall s \in S.$$

The proof of this proposition is straightforward.

In addition to the formulation above, a variation of Eq. (2) can also be considered. For this, assume that every state is reachable from the initial state σ. Then, the bounds can be written for $\forall j \in A$, $\forall i = 1 \dots n_{j-1}$ as:

$$\begin{aligned} \theta(s_{i+1}^j) \leq \ &\gamma\theta(s_i^j) - r(s_i^j, a_i^j) \\ \theta(\sigma) \leq &v^*(\sigma). \end{aligned} \tag{6}$$

Unlike Eq. (2), these constraints alone do not guarantee that the function θ will be a lower bound on the optimal value of the states. Figure 2 depicts a situation in which these bounds are satisfied, but there is a feasible solution that is not an upper bound. Similarly, as in Eq. (2), the bounds may be formulated as:

$$\begin{aligned} (I - \gamma T_a)My \geq r_a \quad \forall a \in A \\ \theta(\sigma) \leq v(\sigma) \end{aligned}$$

We denote this feasible set G_2, and thus $\theta \in G_2$.

Using the upper and lower bounds, we can formulate a new linear program to solve the approximation problem:

$$\begin{aligned} &\text{minimize} \quad \delta \\ &\text{subject to} \quad h(s) - \theta(s) \leq \delta \quad \forall s \in S \\ &\qquad\qquad h \in H_3 \quad \theta \in G_1 \end{aligned} \tag{7}$$

This linear program, unlike Eq. (4), minimizes the max-norm error of the approximation. This is because from the definition $h(s) \geq \theta(s)$ for all s. In addition, even when

the linear program is constructed from the samples only, this inequality holds. Notice that the number of constraints $h(s)-\theta(s) \leq \delta$ is too large, because one constraint is needed for each state. Therefore, in practice these constraints will be sampled as well as the remaining states. In particular, we use those states s for which $v^*(s)$ is known. While it is possible to use G_2 instead of G_1, that somewhat complicates the analysis. We summarize below the main reasons why the formulation in Eq. (7) is more suitable than Eq. (4).

Approximation Bounds

We showed above how to formulate the linear programs for optimizing the heuristic function. It is however important whether these linear programs are feasible and whether their solutions are close to the best heuristic that can be represented using the features in basis M. In this section, we extend the analysis used in approximate linear programming to show new conditions for obtaining a good heuristic function.

We are interested in bounding the maximal approximation error $\|h - v^*\|_\infty$. This bound limits the maximal error in any state, and can be used as a rough measure of the extra search effort required to find the optimal solution. Alternatively, given that $\|h - v^*\|_\infty \leq \epsilon$, then the greedily constructed solution with this heuristic will have the approximation error of at most $m\epsilon$, where m is the number of steps required to reach the goal. This makes it possible to solve the problem without search. For simplicity, we do not address here the issues related to limited sample availability, which have been previously analyzed (de Farias 2002; de Farias and Roy 2004; Ben-Tal and Nemirovski 2008; Goldfarb and Iyengar 2003)

The approximation bound for the solution of Eq. (4) with the constraints in Eq. (2) comes from approximate linear programming (de Farias 2002). Assuming there is a z such that $e = Mz$, the bound is:

$$\|v^* - h\|_c \leq \frac{2}{1-\gamma} \min_x \|v^* - Mx\|_\infty,$$

where $\|\cdot\|_c$ is an L_1 error bound weighted by a vector c, elements of which sum to 1. The approximation bound contains the multiplicative factors, because even when Mx is close to v^* it may not satisfy the required feasibility conditions. This bound only ensures that the sum of the errors is small, but errors in some of the states may still be very large. The bound can be directly translated to an L_∞ bound, assuming that $c = e$, that is a vector of all ones. The bound is as follows:

$$\|v^* - h\|_\infty \leq |\mathcal{S}| \frac{2}{1-\gamma} \min_x \|v^* - Mx\|_\infty.$$

The potential problem with this formulation is that it may be very loose when: (1) the number of states is large, since it depends on the number of states $|\mathcal{S}|$; or (2) the discount factor γ is close to 1 or is 1.

We show below how to address these problems using the alternative formulation of Eq. (7) and taking advantage of additional structure of the approximation space. In the following, we use e to denote the vector of all ones.

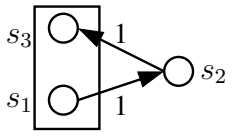

Figure 3: An approximation with loose bounds.

Lemma 2. *Assuming that $e = Mz$ for some z. Then there exists a heuristic function $\hat{h}$ that is feasible in Eq. (3) and satisfies:*

$$\|\hat{h} - v^*\|_\infty \leq \frac{2}{1-\gamma\alpha} \min_x \|v^* - Mx\|_\infty.$$

The proof is similar to that of Lemma 4. Using similar analysis, the following lemma can be shown.

Lemma 3. *Assume that $e = Mz$ for some z. Then there exists a lower bound $\hat{\theta}$ that is feasible in Eq. (5), such that:*

$$\|\hat{\theta} - v^*\|_\infty \leq 2 \min_x \|v^* - Mx\|_\infty.$$

This lemma can be proved simply by subtracting ϵe from θ that is closest to v^*. The above lemmas lead to the following theorem with respect to the formulation in Eq. (7).

Theorem 1. *Assume that $e = Mz$ for some z, and let $\hat{h}, \hat{\theta}, \delta$ be an optimal solution of Eq. (7). Then:*

$$\delta = \|\hat{h} - v^*\|_\infty \leq \left(2 + \frac{2}{1-\gamma\alpha}\right) \min_x \|v^* - Mx\|_\infty.$$

Proof. Assume that that the solution δ does not satisfy the inequality. Then, using Lemma 3 and Lemma 2, it is possible to construct a solution $\hat{h}, \hat{\theta}, \hat{\delta}$. This leads to a contradiction, because $\hat{\delta} < \delta$. □

Therefore, by solving Eq. (7) instead of Eq. (4), the error is independent of the number of states. This is a significant difference, since the approach is proposed for problems with a very large number of states.

Even when Eq. (7) is solved, the approximation error depends on the factor $1/(1-\gamma\alpha)$. For $\gamma = \alpha = 1$, the bound is infinite. In fact the approximate linear program may become infeasible in this case, unless the approximation basis M satisfies some requirements. In the following, we show which requirements are necessary to ensure that there will always be a feasible solution.

To illustrate this problem with the approximation, consider the following simple example with states $\mathcal{S} = \{s_1, s_2, s_3\}$ and a single action $\mathcal{A} = \{a\}$. The goal is the state $\tau = s_3$, and thus there is no transition from this state. The transitions are $t(s_i, a) = s_{i+1}$, for $i = 1, 2$. The rewards are also $r(s_i, a) = 1$ for $i = 1, 2$. Now, let the approximation basis be $M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}^T$. This example is depicted in Figure 3, in which the square represents the aggregated states in which the heuristic function is constant. The bounds of Eq. (2) in this example are

$$\begin{aligned} h(s_1) &\geq \gamma h(s_2) + 1 \\ h(s_2) &\geq \gamma h(s_3) + 1 \\ h(s_3) &\geq 0 \end{aligned}$$

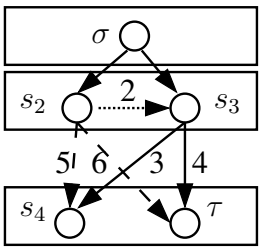

Figure 4: An example of the Lyapunov hierarchy. The dotted line represents a constraint that needs to be removed and replaced by the dashed ones.

The approximation basis M requires that $h(s_1) = h(s_3)$. Thus we get that:

$$h(s_1) \geq \gamma h(s_2) + 1 \geq \gamma^2 h(s_3) + \gamma + 1 = \gamma^2 h(s_1) + \gamma + 1.$$

As a result, despite the fact that $v(s_1) = 2$, the heuristic function is $h(s_2) = (1+\gamma)/(1-\gamma^2)$. This is very imprecise for high values of γ. A similar problem was addressed in standard approximate linear programming by introducing so called Lyapunov vectors. We build on this idea to define conditions that enable us to use Eq. (3) with high γ and α.

Definition 2 (Lyapunov vector hierarchy)**.** Let $u^1 \dots u^k \geq 0$ be a set of vectors, and T and r be partitioned into T_i and r_i respectively. This set of vectors is called a Lyapunov vector hierarchy if there exist $\beta_i < 1$ such that:

$$\begin{aligned} T_i u^i &\leq \beta_i u^i \\ T_j u^i &\leq 0 \quad \forall j < i \end{aligned}$$

The second condition requires that no states in partition j transit to a state with positive u^i.

An example of such a hierarchy would be an abstraction, depicted in Figure 4. Let the state space $\mathcal{S}$ be partitioned into l subsets $\mathcal{S}_i$, with $i = 1 \dots l$. Assume that the transitions satisfy:

$$\forall a \; t(s,a) = s' \wedge s \in \mathcal{S}_i \wedge s' \in \mathcal{S}_j \Rightarrow j < i.$$

That is, there is an ordering of the partitions consistent with the transitions. Let u^i be a vector of the size of the state space, defined as:

$$u^i(k) = 1 \Leftrightarrow s_k \in \mathcal{S}_i,$$

and zero otherwise. It is easy to show that these vectors satisfy the requirements of Definition 2. When the approximation basis M can be shown to contain such u^i, it is, as we show below, possible to use the formulation with $\gamma = \alpha = 1$ with low approximation error.

Lemma 4. *Assume that there exists a Lyapunov hierarchy $u^1 \dots u^l$, and for each u^i there exists z_i such that $u^i = Mz_i$. Then there exists a heuristic function $\hat{h}$ in M that is feasible in Eq. (3), such that:*

$$\|\hat{h} - v^*\|_\infty \leq \prod_{i=1}^{l} \left(\frac{(1+\alpha\gamma)\max_k u^i(k)}{(1-\alpha\gamma\beta_i)\min_k u_i^i(k)} \right) 2 \min_x \|v^* - Mx\|_\infty,$$

where u_i^i is the vector u^i restricted to states in partition i.

Proof. First, let

$$\epsilon = 2\|\tilde{h}_1 - v^*\|_\infty = 2 \min_x \|Mx - v^*\|_\infty.$$

Construct $\tilde{h} = \tilde{h}_1 + \epsilon e$ such that:

$$v^* \leq \tilde{h} \leq v^* + \epsilon.$$

The proof follows by induction on the size l of the Lyapunov hierarchy. Assume that the inequalities are satisfied for all $i' < i$, with the error ϵ and the property that the current $\tilde{h} \geq v^*$. Then let $\hat{h} = \tilde{h} + de$, for some d. Then, using Lemma 1, we have:

$$\begin{aligned} \hat{h} &= \tilde{h} + du^i \geq v^* + du^i \geq T_i v^* + r_d u^i \\ &\geq T_i \tilde{h} - \gamma\alpha\epsilon e + r + du^i \\ &\geq T_i(\hat{h} - du^i) - \gamma\alpha\epsilon e + r + du^i \\ &\geq T_i \hat{h} + r - \alpha\beta_i\gamma du^i + du^i - \gamma\alpha\epsilon e \end{aligned}$$

To satisfy $\hat{h} \geq T_i \hat{h} + r_i$, set d to:

$$\begin{aligned} \alpha\beta_i\gamma du^i + du^i &\geq \gamma\alpha\epsilon e \\ d &\geq \frac{\gamma\alpha}{1-\alpha\beta_i\gamma} \frac{1}{\min_k u_i^i(k)} \epsilon. \end{aligned}$$

Therefore the total approximation error for $\hat{h}$ is:

$$\|\hat{h} - v^*\|_\infty \leq \frac{\gamma\alpha}{1-\alpha\beta_i\gamma} \frac{\max_k u^i(k)}{\min_k u_i^i(k)} \epsilon$$

The lemma follows because $d \geq 0$ and $u^i \geq 0$, and thus the condition $\tilde{h} \geq v^*$ is not violated. In the end, all the constraints are satisfied from the definition of the Lyapunov hierarchy. □

The bound on the approximation error of the optimal solution of Eq. (7) may be then restated as follows.

Theorem 2. *Assume that there exists a Lyapunov hierarchy $u^1 \dots u^l$, and for each u^i there exists z_i such that $u^i = Mz_i$. Then for the optimal solution $\hat{h}, \delta$ of Eq. (7):*

$$\delta = \|\hat{h} - v^*\|_\infty \leq \left(1 + \prod_{i=1}^{l} \frac{(1+\alpha\gamma)\max_k u^i(k)}{(1-\alpha\gamma\beta_i)\min_k u_i^i(k)} \right) 2\epsilon,$$

where $\epsilon = \min_x \|v^* - Mx\|_\infty$.

The proof follows from Lemma 4 similarly to Theorem 1. The theorem shows that even when $\gamma = 1$, it is possible to guarantee the feasibility of Eq. (7) by including the Lyapunov hierarchy in the basis.

A simple instance of a Lyapunov hierarchy is a set of features that depends on the number of steps from the goal. Therefore, the basis M must contain a vector u^i, such that $u^i(j) = 1$ if the number of steps to get to s_j is i and 0 otherwise. This is practical in problems in which the number of steps to the goal is known in any state. Assuming this simplified condition, Theorem 2, may be restated as follows.

$$\|\hat{h} - v^*\|_\infty \leq \left(1 + \prod_{i=1}^{l} \frac{1+\alpha\gamma}{1-\alpha\gamma\beta_i} \right) 2 \min_x \|v^* - Mx\|_\infty.$$

This however indicates an exponential growth in error with the size of the hierarchy with $\gamma = \alpha = 1$. Unfortunately, it is possible to construct an example in which this bound is tight. We have not observed such behavior in the experiments, and it is likely that finer error bounds could be established. As a result of the analysis above, if the basis contains the Lyapunov hierarchy, the approximation error is finite even for $\gamma = 1$.

In some problems it is hard to construct a basis that contains a Lyapunov hierarchy. An alternative approach is to include only constraints that obey the Lyapunov hierarchy present in the basis. These may include multi-step constraints, as indicated in Figure 4. As a result, only a subset of the constrains is added, but this may improve the approximation error significantly. Another option is to define features that depend on the actions, not only on states. This is a non-trivial extension, however, and we leave the details to future work. Finally, when all the rewards are negative and the basis contains only a Lyapunov hierarchy, then it can be shown that no constraints need to be removed.

Experiments

We evaluate the approach on the sliding eight tile puzzle problem–a classic search problem (Reinefeld 1993). The purpose of these experiments is to demonstrate the applicability of the proposed approach. We used the eight-puzzle particularly because it has been studied extensively and because it can be solved relatively quickly. This allows us to evaluate the quality of the heuristic functions we obtain in different settings. Since all the experiments we describe took less than a few seconds, scaling to large problems with many sample plans is very promising. Scalability mostly relies on the ability to solve efficiently large linear programs–an area that has seen significant progress over the years. In all instances, we use the formulation depicted by Eq. (7) with different values of α.

Our basis construction method relies on a set of features available for the domain. Good features are crucial for obtaining a useful heuristic function, since they must be able to discriminate states based on their heuristic value. In addition, the set of features must be limited to facilitate generalization. Notice that although the features are crucial, they are in general much easier to select compared with a good admissible heuristic function. We consider the following basis choices:

1. Manhattan distance of each tile from its goal position, including the empty tile. This results in 9 features that range in values from 0 to 4. This basis does not satisfy the Lyapunov property condition. The minimal admissible heuristic function from these features will assign value -1 to the feature that corresponds to each tile, except the empty tile, which is 0.
2. Abstraction based on the sum of the Manhattan distances of all pieces. For example, feature 7 will be 1 if the sum of the Manhattan distances of the pieces is 7 and 0 otherwise. This basis satisfies the Lyapunov hierarchy condition as defined above, since all the rewards in the domain are negative.

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	States
0	-1	-1	-1	-1	0	0	-1	0	958
0	-1	-1	-1	-1	-1	0	-1	0	70264
0	-1	-1	0	-1	-1	-1	-1	-1	63
0	-1	-1	-1	-1	-1	-1	-1	-1	162

Figure 5: Weights calculated for individual features using the first basis choice. Column x_i corresponds to the weight assigned to feature associated with tile i, where 0 is the empty tile. The top 2 rows are based on data from blind search, and the bottom 2 on data from search based on the heuristic from the previous row.

3. The first feature is the Nilsson sequence score (Nilsson 1971) and the second feature is the total Manhattan distance minus 3. The Nilsson sequence score is obtained by checking around the non-central square 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. This value is not an admissible heuristic.

First, we evaluate the approach in terms of the number of samples that are required to learn a good heuristic function. We first collect samples from a blind search. To prevent expanding too many states, we start with initial states that are close to the goal. This is done by starting with the goal state and performing a sequence of 20 random actions. Typical results obtained in these experiments are shown in Figure 5, all performed with $\alpha = 1$. The column labeled "States" shows the total number of node-action pairs expanded and used to learn the heuristic function, not necessarily unique ones. The samples are gathered from solving the problem optimally for two states. The results show that relatively few nodes are required to obtain a heuristic function that is admissible, and very close to the optimal heuristic function with the given features. Notice that the heuristic function was obtained with no prior knowledge of the domain and without any a priori heuristic function. Very similar results were obtained with the second basis.

Next, we compare the two formulations for the upper bounds, Eq. (1) and Eq. (2), with regard to their approximation error. Notice that a big advantage of the formulation depicted by Eq. (2) is the ability to use transitions from states that are not on a path to the goal. The data is based on 100 goal terminated searches and 1000 additional randomly chosen states. The results are shown in Figure 6. Here, δ is the objective value of Eq. (7), which is the maximal overestimation of the heuristic function in the given samples. Similarly, δ' is the maximal overestimation obtained based on 1000 state samples independent of the linear program. The approximate fraction of states in which the heuristic function *is* admissible is denoted by p. These results demonstrate the tradeoff that α offers. A lower value of α generally leads to a better heuristic function, but at the expense of admissibility. The bounds with regard to the sampled constraints, as presented in (de Farias 2002) do not distinguish between the two formulations. A deeper analysis of this is an important part of future work. Interestingly, the results show that the Nilsson sequence score is not admissible, but it becomes

α	1	0.99	0.9	0.8	0
x_1	0	-0.26	-1	-1	-1
x_2	-0.25	-0.25	-0.59	-0.6	-0.6
x_3	0	0	2.52	2.6	2.6
δ	17	16.49	13.65	13.6	13.6
p	1	1	0.97	0.96	0.95
δ'	19	17	15	14.4	14.4

Figure 6: The discovered heuristic functions as a function of α in the third basis choice, where x_i are the weights on the corresponding features in the order they are defined.

admissible when divided by 4.

We also applied the proposed approach to Tetris, a popular benchmark problem in reinforcement learning (Szita and Lorincz 2006). Because it is an inherently stochastic problem, we used the Regret algorithm (Mercier and Hentenryck 2007) to solve it using a deterministic method. The Regret algorithm first generates samples of the uncertain component of the problem and then treats it as a deterministic problem. It is crucial in Tetris to have a basis that contains a Lyapunov hierarchy, since the rewards are positive. Since we always solve the problem for a fixed number of steps forward, such a hierarchy can be defined based on the number of steps remaining as in Figure 4. Our initial experiments with Tetris produced promising results. However, to outperform the state-of-the-art methods–such as approximate linear programming (Farias and Roy 2006) and cross-entropy methods (Szita and Lorincz 2006)–we need to scale up our implementation to handle millions of samples. This is mostly a technical challenge that can be addressed using methods developed by Farias and Roy (2006).

Conclusion

We present a new approach for automatic construction of heuristic functions by extending the applicability of some common reinforcement learning techniques to this problem. When applied naively, these techniques may lead to formulations that have no feasible solutions. We showed that by guaranteeing certain properties, it is possible to ensure that the approximation is finite and–in some cases–accurate, with tight error bounds. This work lays the foundation for further understanding of how sampling techniques can produce good heuristic functions for complex planning problems.

Learning heuristic functions automatically is a longstanding challenge in artificial intelligence. Despite the existing shortcomings, the new approach we propose has several important advantages. The formulation is very general and it places only modest assumptions on the features. It requires little domain knowledge. It works well with samples of plans–both optimal and non-optimal ones. And, most importantly, it makes it possible to compute guarantees on the admissibility of the learned heuristic function.

Acknowledgements

This work was supported in part by the Air Force Office of Scientific Research under Grant No. FA9550-08-1-0171 and by the National Science Foundation under Grant No. IIS-0535061. The findings and views expressed in this paper are those of the authors and do not necessarily reflect the positions of the sponsors.

References

Beliaeva, N., and Zilberstein, S. 2005. Generating admissible heuristics by abstraction for search in stochastic domains. In *Abstraction, Reformulation and Approximation*. Springer Berlin / Heidelberg. 14–29.

Ben-Tal, A., and Nemirovski, A. 2008. Selected topics in robust optimization. *Mathematical Programming, Series B* 112:125–158.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence* 129(1-2):5–33.

Bylander, T. 1997. A linear programming heuristic for optimal planning. In *National Conference on Artificial Intelligence*, 694–699.

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. In *Advances in Artifical Intelligence*. Springer Berlin / Heidelberg. 402–416.

de Farias, D. P., and Roy, B. V. 2004. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research* 29(3):462–478.

de Farias, D. P. 2002. *The Linear Programming Approach to Approximate Dynamic Programming: Theory and Application*. Ph.D. Dissertation, Stanford University.

Farias, V., and Roy, B. V. 2006. *Probabilistic and Randomized Methods for Design Under Uncertainty*. Springer-Verlag. chapter 6: Tetris: A Study of Randomized Constraint Sampling.

Goldfarb, D., and Iyengar, G. 2003. Robust convex quadratically constrained programs. *Mathematical Programming* 97:495–515.

Hansen, E. A., and Zilberstein, S. 2001. LAO *: A heuristic search algorithm that finds solutions with loops. *Artificial Intelligence* 129(1-2):35–62.

Hoffman, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Aritifial Intelligence Research* 14:253–302.

Holte, R. C.; Mkadmi, T.; Zimmer, R.; and MacDonald, A. 1996. Speeding up problem solving by abstraction: a graph oriented approach. *Artificial Intelligence* 85:321–361.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. In *Abstraction, Reformulation and Approximation*. Springer Berlin / Heidelberg. 121–133.

Mercier, L., and Hentenryck, P. V. 2007. Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. In *International Joint Conference on AI*, 1979–1985.

Nilsson, N. 1971. *Problem-Solving Methods in Artificial Intelligence*. McGraw Hill.

Powell, W. B. 2007. *Approximate Dynamic Programming*. Wiley-Interscience.

Reinefeld, A. 1993. Complete solution of the eight-puzzle and the benefit of node ordering in IDA*. In *International Joint Conference on AI*, 248–253.

Szita, I., and Lorincz, A. 2006. Learning Tetris using the noisy cross-entropy method. *Neural Computation* 18(12):2936–2941.

Trick, M. A., and Zin, S. E. 2005. Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics* 1(1):255–277.