Toward Discriminative Learning of Planning Heuristics

Yuehua Xu and **Alan Fern**
School of EECS, Oregon State University

Abstract

We consider the problem of learning heuristics for controlling forward state-space search in AI planning domain. We draw on a recent framework for "structured output classification" (e.g. syntactic parsing) known as learning as search optimization (LaSO). The LaSO approach uses discriminative learning to optimize heuristic functions for search-based computation of structured outputs and has shown promising results in a number of domains. However, the search problems that arise in AI planning tend to be qualitatively very different from those considered in structured classification, which raises a number of potential difficulties in directly applying LaSO to planning. In this paper, we discuss these issues and describe a LaSO-based approach for discriminative learning of beam-search heuristics in AI planning domains. Our preliminary results in three benchmark domains are promising. In particular, across a range of beam-sizes the discriminatively trained heuristic outperforms the one used by the planner FF and another recent non-discriminative learning approach.

Introduction

It is somewhat surprising that a number of today's state-of-the-art planners are based on the old idea of forward state-space heuristic search (Bonet & Geffner 1999; Hoffmann & Nebel 2001; Nguyen, Kambhampati, & Nigenda 2002). These performances are primarily due to the recent progress in defining domain-independent heuristic functions that provide good guidance across a wide range of domains. However, there remain many domains where these heuristics are deficient, leading to planning failure. One way to improve the applicability and robustness of such planning systems is to develop learning mechanisms that automatically tune the heuristic to a particular domain based on prior planning experience. In this work, we consider the applicability of recent developments in machine learning to this problem. In particular, given a set of solved planning problems from a target domain, we consider using discriminative learning techniques for acquiring a domain-specific heuristic for controlling beam search.

Despite the potential benefits of learning to improve forward state-space planning heuristics, there have been few reported successes. While there has been a substantial body of work on learning heuristics or value functions to control search, e.g. (Boyan & Moore 2000; Zhang & Dietterich 1995; Buro 1998), virtually all such work has focused on search optimization problems. These problems involve finding a "least cost" configuration of some combinatorial object and have a much different flavor than the types of domains encountered in benchmarks from AI planning. To our knowledge, no such previous system has been demonstrated on benchmark domains from AI planning.

Recent work (Yoon, Fern, & Givan 2006) has made progress toward learning heuristics for planning domains. The work focused on improving the heuristic used by the state-of-the-art planner FF (Hoffmann & Nebel 2001). In particular, the approach used linear regression to learn an approximation of the difference between FF's heuristic and the observed distances-to-goal of states in the training plans. The sum of FF's heuristic and the learned regression function was then used as the new domain-specific heuristic. The primary contribution of the work was to define a generic knowledge representation for features and a features-search procedure that allowed learning of good regression functions across a range of planning domains. While the approach showed promising results, the learning mechanism has a number of potential shortcomings. Most importantly, the mechanism does not consider the actual search performance of the heuristic during learning. That is, learning is based purely on approximating the observed distances-to-goal in the training data. Even if the learned heuristic performs poorly when used for search in one or more training problems, no attempt is made to correct the heuristic in response.

In this paper, we consider a learning approach that tightly couples learning with the actual search procedure, iteratively updating the heuristic in response to observed search errors. This approach is discriminative in the sense that it only attempts to learn a heuristic that discriminates between "good" and "bad" states well enough to find the goal, rather than attempting to precisely model the distance-to-goal. In many areas of machine learning, such discriminative methods have been observed to outperform their non-discriminative counterparts. A main goal of this work is to demonstrate such benefits in the context of planning.

Our learning approach is based on the recent framework of learning as search optimization (LaSO) (Daume III & Marcu 2005), which was developed to solve "structured output classification" problems. Such problems involve mapping structured inputs (e.g. sentences) to structured outputs (e.g. syntactic parses) and classification can be posed as performing a search over candidate outputs guided by a heuristic. LaSO provides an approach for discriminative learning of such heuristics and has demonstrated good performance across several structured classification problems. However, the search problems corresponding to structured classification are qualitatively very different from those typical of AI

planning domains. For example, in structured classification the search problems typically have a single or small number of solution paths, whereas in AI planning there are often a very large number of equally good solutions. Given these differences, the utility of LaSO in our context is not clear.

The main contribution of this paper is to describe a LaSO-inspired approach to learn beam-search heuristics for AI planning and to provide a preliminary evaluation. Our empirical results show that the approach is able to learn heuristics that improve beam-search compared to using the heuristic from the planner FF. In addition, the results show that discriminative learning appears to have an advantage over the existing non-discriminative approach.

Below, we first give our problem setup for learning planning heuristics. Next, we give an overview of the LaSO framework for structured classification. This is followed by a description of our LaSO-based approach to learning beam-search heuristics for planning domains, along with an analysis of convergence. Finally, we present experimental results and conclude with a discussion of future directions.

Learning Planning Heuristics

Planning Domains. A planning domain $\mathcal{D}$ defines a set of possible actions $\mathcal{A}$ and a set of states $\mathcal{S}$ in terms of a set of predicate symbols P, action types Y, and constants C. A state fact is the application of a predicate to the appropriate number of constants, with a state being a set of state facts. Each action $a \in \mathcal{A}$ consists of: 1) an action name, which is an action type applied to the appropriate number of constants, 2) a set of precondition state facts $\text{Pre}(a)$, 3) two sets of state facts $\text{Add}(a)$ and $\text{Del}(a)$ representing the add and delete effects respectively. As usual, an action a is applicable to a state s iff $\text{Pre}(a) \subseteq s$, and the application of an (applicable) action a to s results in the new state $s' = (s \setminus \text{Del}(a)) \cup \text{Add}(a)$.

Given a planning domain, a planning problem is a tuple (s, A, g), where $A \subseteq \mathcal{A}$ is a set of actions, $s \in \mathcal{S}$ is the initial state, and g is a set of state facts representing the goal. A solution plan for a planning problem is a sequence of actions $(a_1, \ldots, a_l)$, where the sequential application of the sequence starting in state s leads to a goal state s' where $g \subseteq s'$. In this paper, we will view planning problems as directed graphs where the vertices represent states and the edges represent possible state transitions. Planning then reduces to graph search for a path from the initial state to goal.

Learning to Plan. Planning competitions typically include many planning domains, and each one provides a sequence of planning problems, often in increasing order of difficulty. Despite the fact that the planners in these competitions experience many problems from the same domain, to our knowledge none of them have made any attempt to learn from previous experience in a domain. Rather they solve each problem as if it were the first time the domain had been encountered. The ability to effectively transfer domain experience from one problem to the next would provide a tremendous advantage. However, to date, "learning to plan" systems have lagged behind the state-of-the-art in non-learning domain-independent planners. The motivation of this work is to move toward reversing that trend.

In particular, here we focus on developing learning capabilities within the simple, but highly successful, framework of heuristic state-space search planning. Our goal is to learn heuristics that can quickly solve problems using beam search with a small beam width. Given a representative training set of problems from a planning domain, our approach first solves the problems using potentially expensive search (e.g. using a large beam width), guided by an existing heuristic. These solutions are then used to learn a heuristic that can guide a small width beam search to the same solutions. The hope is that the learned heuristic will then generalize and allow for the quick solution of new problems that could not be practically solved before learning.

Heuristic Representation. In this work we will consider learning heuristic functions that are represented as weighted linear combinations of features, i.e. $H(n) = \Sigma_i w_i \cdot f_i(n)$ where n is a search node, f_i is a feature of search nodes, and w_i is the weight (importance) of feature f_i. In particular, for each domain we would like to learn a distinct set of features and their corresponding weights that lead to good planning performance in that domain. One of the challenges with this approach is to define a generic feature space from which features are selected. This space must be rich enough to capture important properties of a wide range of planning domains, but also be amenable to searching for those properties. For this purpose we will draw on prior work (Yoon, Fern, & Givan 2006) that defined such a feature space, based on properties of relaxed plans, and described a search approach for finding useful features. In this investigation, we will use the features from that work in addition to using the relaxed-plan length heuristic of FF plan as a feature.

The approach of (Yoon, Fern, & Givan 2006) used a very simple weight learning method, where weights were tuned by linear regression to predict the distance-to-goal of search nodes in the training set. While this approach showed promise, it is oblivious to the actual performance of the heuristic when used for search. In particular, even if the heuristic provides poor guidance for search on the training problems no further learning will occur. The main objective of this work is to improve performance by investigating a more sophisticated weight learning mechanism that is tightly integrated with the search process, iteratively adapting the heuristic in response to observed search errors. Below we first describe prior work from structured classification upon which our approach is based, and then describe its adaptation to our setting.

Learning Heuristics for Structured Classification

Structured classification is the problem of learning a mapping from structured inputs to structured outputs. A prototypical example is the problem of part-of-speech tagging where the goal is to learn a mapping from word sequences (i.e. sentences) to sequences of part-of-speech tags that correctly label each word. There has been much recent progress in structured classification including methods based on condition random fields (Lafferty, McCallum, & Pereira 2001), Perceptron updates (Collins 2002),

and margin optimization (Taskar, Guestrin, & Koller 2003; Tsochantaridis *et al.* 2004).

A recent alternative approach (Daume III & Marcu 2005) views structured classification as a search problem and learns by optimizing a heuristic for that problem based on training data. In particular, given a structured input x, the problem of labeling x by a structured output y is treated as searching through an exponentially large set of candidate outputs. For example, in part-of-speech tagging where x is a sequence of words and y a sequence of tags that label the words of x, each node in the search space is a pair (x, y') where y' is a partial labeling of the words in x. Learning corresponds to inducing a heuristic that quickly directs search to the search node (x, y) where y is the desired output. This framework, known as *learning as search optimization (LaSO)*, has demonstrated state-of-the-art performance on a number of structured classification problems and serves as the basis for our work.

We note that the original presentation of LaSO (Daume III & Marcu 2005) formulated the problem as one of learning the cost function $G(n)$ over search nodes (i.e. distance from the start node) rather than learning a heuristic $H(n)$. However, it turns out that this naming convention was somewhat arbitrary, as no particular constraints were imposed on the learned function. Thus, the LaSO algorithm can be viewed as learning heuristic functions, which is the view that we will follow for the remainder of this paper.

LaSO assumes the availability of a feature vector $F(n) = \langle f_1(n), \ldots, f_m(n) \rangle$ that is a function of search node n. These features are intended to capture properties of search nodes that might be useful in evaluating their quality. For example, in part-of-speech tagging, it is typical to have indicator features that detect when particular words are labeled by certain tags and also features about the sequential tag structure, e.g. that count the number of times an article-tag was followed by a noun-tag in a partial labeling y'. The heuristic is assumed to be a linear combination of these features $H(n) = F(n) \cdot w$, where w is a vector of feature weights. Thus, the main goal of LaSO is to optimize w so that the learned heuristic is maximally effective.

A key feature of LaSO is that it integrates learning into the search process, updating the weights whenever a search error is observed. In order to determine when errors occur, LaSO assumes the availability of a function *solvable(n)*, which returns true if and only if the search node n is on a path to the desired goal node (x, y). For example, in part-of-speech tagging, if we assume that the search operators are only able to add labels to currently unlabeled words, then *solvable*$((x, y'))$ will be true iff the partial labeling y' can be completed to arrive at the goal output y.

Figure 1 shows the generic integrated learning-search procedure of LaSO. This procedure can capture various search strategies, e.g. best-first search and certain types of beam search by altering the *Sort* procedure (e.g. only keeping the top b nodes in the queue). The function *CandidateTest(n)* returns true if node n is a possible goal node, i.e. a node that corresponds to a complete (but possibly incorrect) structured output y. For example, in part-of-speech tagging, this test would return true for any search node that assigns a part-of-speech tag to each word of the input sequence x. Given an input x the heuristically guided search terminates when a node is dequeued that passes the candidate test and returns the corresponding structured output. Thus, LaSO's learning objective is to find a set of weights such that, for each training example, the first dequeued node that passes the candidate test corresponds to the desired target output y.

The LaSO procedure takes as input a single training example (x, y) along with the current weight vector and returns an updated weight vector. This procedure is used by cycling through the training examples and iteratively applying LaSO to each one until the learned heuristic is able to correctly classify each weight, or a iteration limit is reached. On each call LaSO begins a search guided by the current heuristic starting from the initial node $(x, \textbf{null})$. Whenever an error is made the weights are updated in order to help avoid the same error in the future.

There are two possible error conditions as shown in the first **if** statement: 1) the current queue does not contain any solvable nodes, as determined by *solvable((x,y'))*, and hence further search is fruitless, and 2) the most recently dequeued node passes the candidate test but does not correspond to the desired goal node, meaning that the search would have terminated with an incorrect output. In either case, the procedure then computes the set of solvable sibling nodes of n using the procedure *Siblings*$((x, y'))$. The weights are then updated so that they will tend to assign a better heuristic value to the sibling nodes rather than the nodes currently in the queue. By doing this, the hope is that next time through the search it will be more likely that the heuristic will rank the siblings higher than the unsolvable nodes, as desired. In the next section, we will give the Perceptron-based weight updating rule used in our implementation. Finally, after the weight update, the queue is set equal to the set of siblings and the search continues. Note, that when no errors are made the procedure operates as an ordinary search procedure, terminating with the desired output.

For certain types of weight updates, (Daume III & Marcu 2005) states convergence results that bound the number of mistakes LaSO will make on the training data under certain assumptions.

LaSO $((x, y), w)$
$Q \leftarrow \{(x, \textbf{null})\}$ // initialize queue
while Q is not empty **do**
  $(x, y') \leftarrow$ ReturnFirst(Q)
  if (no node in Q is solvable) **or**
    (CandidateTest((x, y')) and $\neg$*solvable*$((x, y'))$) **then**
      $S \leftarrow$ Siblings((x, y'))
      $w \leftarrow$ Update(w, S, Q)
      $Q \leftarrow S$
  else
      if CandidateTest((x, y')) **then** return w
      next $\leftarrow$ Expand((x, y'))
      $Q \leftarrow$ Sort($(Q \setminus (x, y')) \cup$ next)
  end if
end while

Figure 1: The LaSO algorithm.

Learning Beam-Search Heuristics for Planning

Given the success of LaSO in the context of structured classification, it is interesting to consider its applicability to a wider range of search problems. Here we focus on search in the context of AI planning. Recall that our "learning to plan" training set contains planning problems along with example solutions. We can view this problem as one of structured classification where we are given a training set $\{(x_i, y_i)\}$, where each $x_i = (s_0, g)$ is a planning problem and each $y_i = (s_0, s_1, ..., s_T)$ is a sequence of states that solves the corresponding planning problem. Given this view, we can consider applying LaSO in order to learn a heuristic for controlling forward state-space search. That is, use the LaSO procedure to learn a heuristic that guides search toward the specific plan given by y_i for each x_i.

While in concept it is straightforward to map planning to the LaSO framework, it is not so obvious that the approach will work well. This is because the search problems arising in AI planning have very different characteristics compared to those tackled by LaSO so far. Most notably, there are typically a large number of good (even optimal) solutions to any given planning problem. These solutions may take very different paths to the goal or may result by simply reordering the steps of a particular plan. For example, in the Blocks world, in any particular state, there are generally many possible good next actions as it does not matter which order the various goal towers are constructed. Despite the possibility of many good solutions, LaSO will attempt to learn a heuristic that strictly prefers the training-set solutions over other equally good solutions that are not in the training set. This raises the potential for the learning problem to be impossible to solve or very difficult since many of the other good solutions to x_i may be inherently identical to y_i. In such cases, it is simply not clear whether the weights will converge to a good solution or not.

One approach to overcoming this difficulty might be to include many or all possible solutions in the training set. However, in general, this will be impractical due to the enormous number of good solutions. In this work, we simply note this practical concern and empirically evaluate the approach. Below we describe a specific instantiation of LaSO used for our planning experiments. Our instantiation is not a precise refinement of the original LaSO as described in the previous section. Rather, we use a slightly modified version of the procedure that incorporates a form of beam search that is not captured by the original procedure and that we found more useful in the context of planning. We will refer to the modified procedure as LaSO*.

Beam search. In beam search, a beam B which contains b nodes is generated at each step of the search process, where b is the beam size. At each step, all of the nodes on the current beam are expanded and the top b children, as scored by the heuristic, are taken to be the next beam. This process continues until a goal node appears on the beam, at which point a solution plan has been found. When the beam size is small, many nodes in the search space are pruned away, often resulting in the inability to find a solution or finding very sub-optimal solutions. When the beam size increases, the quality of the solutions tend to improve, however, both the time and space complexity increases linearly with the beam size, leading to practical limitations. The goal of our work is to learn a domain-specific heuristic that allows for beam search with small b to replicate the result of using a large b. This can be viewed as a form of speed-up learning.

Discriminative Learning. The input to our heuristic learner is a set $\{(x_i, y_i)\}$ of pairs, where $x_i = (s_0, g)$ is a training problem from the target planning domain and $y_i = (s_0, s_1, \ldots, s_T)$ is a state sequence corresponding to a solution plan for x_i. Intuitively, our training procedure will attempt to find weights such that for each problem the j'th state in the solution is contained in the j'th beam of the search. A search error is said to have occurred if this is ever not the case. Figure 2 gives pseudo-code for the overall heuristic learning approach. The top-level procedure repeatedly cycles through the training set passing each example to LaSO* to arrive at updated weights. The procedure terminates when the weights remain unchanged after cycling through all examples or a maximum number of iterations is reached.

Given a training example (x, y), LaSO* conducts a beam search starting with the initial beam $\{(x_i, (s_0))\}$, i.e. a single search node with an empty plan. After generating beam j of the search, if $n^* = (x_i, (s_0, s_1, \ldots, s_j))$ is not on the beam then a search error is said to have occurred. In this case, we update the weights in a way that makes n^* more preferred by the heuristic, ideally preferred enough to remain on the beam next time through the search. In this work, we use the Perceptron weight updating rule, which was one of the updates proposed in (Daume III & Marcu 2005)

$$w = w + \alpha \cdot \left(\frac{\sum_{n \in B} F(n)}{|B|} - F(n^*) \right)$$

where $0 < \alpha \leq 1$ is a learning rate parameter, $F(n)$ is the feature vector of search node n and B is the current beam. Intuitively this update rule moves the weights in a direction that decreases the heuristic value (increase the preference) of the desired search node n^* and increases the heuristic value for the nodes in the beam. After the weight update, the beam is replaced by the single search node n^* and the search continues. Note that each call to LaSO* is guaranteed to terminate in T search steps, generating training examples as necessary.

Convergence of LaSO*

We now prove that under certain assumptions LaSO* is guaranteed to converge in a finite number of iterations to a set of weights that solves all of the training examples. The proof is a simple generalization of the one used to prove convergence of Perceptron updates for structured classification (Collins 2002).

Consider a set of training problems (x_i, y_i), where $x_i = (s_0, g)$ and $y_i = (s_0, s_1, \ldots, s_T)$. For each (x_i, y_i) we denote by $n^*_{ij} = (x_i, (s_0, \ldots, s_j))$ the node on the desired search path at depth j for example i. Also let D_{ij} be the set of all nodes that can be reached in j search steps from

HeuristicLearn $(\{(x_i, y_i)\}, b)$
$w \leftarrow 0$
repeat until w is unchanged **or** a large number of iterations
  for every (x_i, y_i)
    LaSO*$((x_i, y_i), w)$
return w

LaSO* $((x, y), w, b)$
// x is a planning problem (s_0, g)
// y is a solution trajectory $(s_0, s_1, \ldots, s_T)$
// w current weight vector
$B \leftarrow \{(x, (s_0))\}$ // initial beam
for $j = 0, \ldots, T-1$
  $B \leftarrow$ **BeamExpand**(B, w, b)
  $n^* \leftarrow (x, (s_1, \ldots, s_{j+1}))$ // desired node
  if $n^* \notin B$ **then**
    $w \leftarrow$ **Update**(w, B, n^*)
    $B \leftarrow \{n^*\}$

BeamExpand (B, w, b)
candidates $\leftarrow \{\}$
for every $n \in B$
  candidates $\leftarrow$ candidates$\cup$ **Successors**(n)
for every $n \in$ candidates
  $H(n) \leftarrow w \cdot F(n)$ // compute heuristic score of n
return b nodes in candidates with lowest heuristic value

Figure 2: The discriminative learning algorithm.

n^*_{i0}. That is, D_{ij} is the set of all possible nodes that could be in the beam after j beam updates. In our result, we will let R be a constant such that

$$\forall i, j, \forall n \in D_{ij}, \|\Phi(n) - \Phi(n^*_{ij})\| \leq R.$$

where $\Phi(n)$ is the feature vector of node n.

As usual for Perceptron-style updates, our results will be stated in terms of the existence of a weight vector that achieves a certain margin on the training set. Here we use a notion of margin that is suited to our beam search framework. As defined below a *beam margin* is a triple (b', δ_1, δ_2) where b' is a non-negative integer, and $\delta_1, \delta_2 \geq 0$.

Definition 1 (Beam Margin)**.** *A weight vector w has beam margin (b', δ_1, δ_2) on a training set $\{(x_i, y_i)\}$ if for each i, j there is a set $D'_{ij} \subseteq D_{ij}$ of size at most b' such that*

$$\forall n \in D_{ij} - D'_{ij} \quad , \quad w \cdot \Phi(n) - w \cdot \Phi(n^*_{ij}) \geq \delta_1 \text{ and,}$$
$$\forall n \in D'_{ij} \quad , \quad \delta_1 > w \cdot \Phi(n) - w \cdot \Phi(n^*_{ij}) \geq -\delta_2$$

According to this definition a weight vector w has beam margin (b', δ_1, δ_2) if at each search depth it ranks the target node n^*_{ij} better than most other nodes by a margin of at least δ_1, and ranks at most b' nodes better than n^*_{ij} by a margin no greater than δ_2. Whenever this condition is satisfied we are guaranteed that a beam search with width $b > b'$ using weights w will solve all of the training problems. The case where $b' = 0$ corresponds to the more typical definition of margin, where the target is required to be ranked higher than all other nodes. By considering the case where $b' > 0$ we can show convergence in cases where no such "dominating" weight vector exists, yet there are weight vectors that allow search to correctly solve the training problems.

Theorem 1. *If there exists a weight vector w, such that $\|w\| = 1$ and w has beam margin (b', δ_1, δ_2) on the training set, then for any beam width $b > \left(1 + \frac{\delta_2}{\delta_1}\right) b'$, the number of mistakes made by LaSO* is bounded by $\left(\frac{bR}{\delta_1(b-b') - \delta_2 b'}\right)^2$.*

The proof is in the appendix. Notice that when $b' = 0$, i.e. there is a dominating weight vector, the mistake bound reduces to $\left(\frac{R}{\delta_1}\right)^2$, which does not depend on the beam width and matches the result stated in (Daume III & Marcu 2005). This is also the behavior when $b >> b'$. In the case when $\delta_1 = \delta_2$ and we use the minimum beam width allowed by the theorem $b = 2b' + 1$, the bound is $\left(\frac{(2b'+1)R}{\delta_1}\right)^2$, which is a factor of $(2b' + 1)^2$ larger than when $b >> b'$. Thus, this result points to a trade-off between the mistake bound and computational complexity of LaSO*. That is, the computational complexity of each iteration increases linearly with the beam width, but the mistake bound decreases as the beam size becomes large. This agrees with the intuition that the more computation time we are willing to put into search at prediction time, the less we need to learn.

Experimental Results

We now present experimental results in three benchmark STRIPS domains: Blocks world, Pipesworld, and PSR. In each domain, we set a time cut-off of 30 CPU minutes and considered a problem to be unsolved if a solution is not found within the cut-off. Given a set of training problems we generated solution trajectories by running both FF and beam search with different beam sizes and then taking the best solution found as the training trajectory. For the blocks world, we used a set of features learned in previous work (Yoon, Fern, & Givan 2005; Fern, Yoon, & Givan 2003) and for Pipesworld and PSR we used the features learned in (Yoon, Fern, & Givan 2006). In all cases, we include the heuristic computed by FF as one of the features.

Our procedure described in the previous section was then used to learn weights for these features using a beam width of 10 and a learning rate of 0.01. In all cases, LaSO* was run for 1000 iterations or until the weights didn't change. The learning times varied across domains, depending on the number of predicates and actions, and the length of solution trajectories. The average time for processing a single problem in a single iteration was about 10 seconds for PSR and less than 1 seconds for Blocks world and Pipesworld.

Domain Details. Our Blocks world problems were generated by the state generator provided by John Slaney. Four operations are used in its representation: **stack** a block onto a block, **unstack** a block from a block, **put down** a block onto the table, and **pick up** a block from the table. Thirty problems, including twenty with 10 blocks and ten with 20 blocks, were used as training data. The testing data includes 10 problems with 20 blocks, 10 problems with 30 blocks, and 10 problems with 40 blocks. There are 15 features in this domain including FF's relax-plan-length heuristic.

Both Pipesworld and PSR are taken from the fourth international planning computation (IPC4). Each domain included 50 problems, roughly ordered by difficulty. We used the first 15 problems for training and the remaining 35 problems for testing. For Pipesworld and PSR there were 35 and 54 features respectively including FF's relaxed-plan-length heuristic.

Performance Across Beam Sizes. Figures 3, 4, and 5 give the performance of beam search in each domain for various beam sizes. The columns correspond to three different algorithms: LEN - beam search using FF's relaxed plan length heuristic, LaSO* - beam search using the heuristic learned using LaSO*, and LR - beam search using the heuristic learned from linear regression as was done in (Yoon, Fern, & Givan 2006). For each algorithm, each row corresponds to a particular beam width and lists the number of solved problems and the averaged plan length of those *solved* problems.

In general, for all algorithms (learning and non-learning) we see that as the beam size increases the number of solved problems increases and solution lengths improve. However, after some point the number of solved problems typically decreases. This behavior is typical for beam search, since as the beam increases there is a greater chance of not pruning a solution trajectory, but the computational time and memory demands increase. Thus, for a fixed time cut-off we expect a decrease in performance. Also note that it is not necessary true that the plan lengths are strictly non-decreasing with beam width. With larger beam widths the number of candidates for the next beam increases, making it more likely for the heuristic to get confused by "bad" states. This is also one possible reason why performance tends to decrease with larger beam sizes.

LaSO* Versus No Learning. Compared to LEN the discriminatively learned heuristic LaSO* tended to significantly improve the performance of beam search, especially for small beam sizes. For example, Figure 3 shows that, with beam size 1, LaSO* solves twice as many problems as LEN in Blocks world. The average plan length has also been reduced significantly, though it is still far from optimal. As the beam size increases the gap between LaSO* and LEN decreases but LaSO* still solves more problems with comparable solution quality. In Pipesworld, Figure 4 shows that LaSO* has the best performance with beam size 5, solving 12 more problems than LEN. As the beam size increases, again the performance gap decreases, but LaSO* consistently solves more problems than LEN. We see that LEN does outperform LaSO* in terms of solution length for large beam sizes. One possible reason for this is that LaSO* solves more problems than LEN and those additional problems may have large solution lengths that are incorporated into the average for LaSO* but not LEN. The trends are similar for the PSR domain, however, for the largest beam sizes both methods solve very few problems with LEN solving slightly more.

All the experimental results show the advantage of LaSO*, solving more problem with smaller beam size than LEN. Because of the smaller beam size, both the time cost and memory cost are reduced significantly.

Comparing LaSO with Linear Regression. To compare with the linear regression approach of (Yoon, Fern, & Givan 2005) we learned weights using linear regression. For every training problem P_i and its solution $T_i = (s_0, s_1, \ldots, s_T)$, we generated a regression training set $(F(s_j), T-j-h(s_j))$ for each $s_j, j = 0, 1, \ldots, T-1$, where $h(s_j)$ is FF's relaxed-plan-length heuristic for state s_j. Here, $T-j$ is the distance from the state s_j to the goal state s_T, computed from the training trajectory. Thus, $T - j - h(s_j)$ is the difference between the actual distance and FF's heuristic. We solve the linear regression problem using the function LinearRegression in the machine learning toolbox Weka. The final linear regression heuristic is a sum of FF's heuristic and the learned function. The results for this procedure are shown in the columns labeled LR.

In Figure 3, LR solves fewer problems than LaSO* with beam sizes smaller than 100 but solves more problems than LaSO* with beam size larger than 100. For Pipesworld, LaSO* always solves more problems than LR. In PSR, Figure 5 shows that LaSO* is better than LR with beam size 5, but becomes slightly worse as the beam size increases.

These preliminary results indicate that discriminative learning can significantly improve over non-discriminative learning and is never much worse. Indeed, there appears to be utility in integrating the learning process directly in the search procedure.

Best First Search results. While our heuristic was learned for the purpose of controlling beam search we conducted one more experiment in each domain where we used the heuristics to guide Best First Search (*BFS*). We include these results primarily because BFS was the search procedure used to evaluate LR in (Yoon, Fern, & Givan 2006). These results are shown in the bottom row of each table.

In Blocks world, LaSO* outperforms the other two algorithms. LaSO* solves 23 problems while FF's relaxed-plan-length heuristic only solves 5 problems. LEN is the best one in Pipesworld and LR works best in PSR. But the performance of LaSO* is very close to the best planner in these two domains. These results indicate that the advantage of the discriminatively learned heuristic over regression is not just restricted to beam search, but appears to extend to other search approaches. Investigating variants of LaSO for BFS and other search strategies is an important future direction.

	Problems solved			Plan length		
b	LEN	LaSO*	LR	LEN	LaSO*	LR
1	13	25	11	6022	2444	4254
5	21	26	19	3094	939	1767
10	22	25	19	2589	1035	368
20	23	27	22	921	671	227
50	20	27	19	522	488	102
100	19	24	20	290	218	157
200	20	21	26	192	241	99
500	17	17	23	101	122	84
1000	16	19	20	100	103	68
BFS	5	23	13	97	167	97

Figure 3: Results on Blocks world.

	Problems solved			Plan length		
b	LEN	LaSO*	LR	LEN	LaSO*	LR
1	11	13	8	375	1739	3888
5	16	28	19	1467	1409	1300
10	17	26	21	2192	740	800
20	17	27	22	161	173	885
50	18	27	21	264	84	4111
100	18	27	21	83	72	165
200	20	26	20	46	74	108
500	21	25	21	39	67	74
1000	20	22	20	31	52	38
BFS	15	11	7	48	46	46

Figure 4: Results on Pipesworld.

	Problems solved			Plan length		
b	LEN	LaSO*	LR	LEN	LaSO*	LR
1	0	0	0	-	-	-
5	0	14	9	-	275	228
10	1	12	13	516	194	160
20	7	12	17	374	183	146
50	13	16	16	154	105	109
100	13	9	13	114	86	86
200	8	5	4	71	73	70
500	4	2	2	55	53	48
1000	1	1	1	39	39	43
BFS	13	18	21	100	119	124

Figure 5: Results on PSR.

Summary and Future Work

This work began with the question of whether the LaSO approach to structured classification could be usefully applied in the context of AI planning. We discussed the qualitative differences between the search problems in AI planning compared to what LaSO has been applied to so far, and the potential difficulties that might arise. Nevertheless, our preliminary investigation in three benchmark planning domains suggests that our LaSO variant is 1) able to significantly improve over the heuristic of FF plan, and 2) improve over the regression based learning method recently proposed in (Yoon, Fern, & Givan 2006). Thus, we conclude that the approach has good promise as a way of learning heuristics to control forward state-space search planners, a problem that to date has received little attention.

Currently our approach assumes a provided set of features. Here we used features that were discovered by previous learning-to-plan research. In future work, we plan to extend the method to allow for feature induction within the learning process. In particular, we plan to develop a feature language that allows us to capture arbitrary constraints on actions and states, compared to the current feature language that only captures properties of states. The constraints can then be combined in a "soft" way by learning weights that combine them appropriately. Prior work on learning to plan (Khardon 1996; Huang, Selman, & Kautz 2000; Fern, Yoon, & Givan 2003) has shown that action constraints (e.g. don't pick-up a "solved" block) are often an effective form of knowledge for planning, while other work demonstrates the potential benefits of state constraints (e.g. the number of solved blocks should monotonically increase) (Yoon, Fern, & Givan 2005). We believe that the LaSO approach will provide a principled framework for learning both types of constraints and learning to weigh them against each other.

Another important direction of future work is to investigate the sensitivity of the LaSO approach to the particular solutions provided in the training data. In addition, understanding more general conditions under which the approach is guaranteed to converge is of interest. Currently the notion of beam margin provides a sufficient but not necessary condition for a set of weights to successfully solve the training problems. That is, there are cases where a set of weights exists that solve the training problems, yet our mistake bound does not apply. The question of convergence in such cases is left as future work.

Also of interest is to investigate the use of plan analysis in LaSO. For example, one might use plan analysis to convert the totally ordered plans in the training data to partially-order plans. LaSO could then be modified so that it only considers a search step to be in error if it violates the partial ordering. We also plan to consider learning heuristics for other search strategies. Of particular interest to us is the integration of beam search and limited discrepancy backtracking as described in (Furcy & Koenig 2005). Finally, applying the framework to other search spaces in planning, such as partial order planning, and more general planning formulations, e.g. probabilistic planning, are important directions.

Acknowledgements

We would like to thank Sungwook Yoon for helpful discussions and for providing critical infrastructure. This work was supported by an NSF award IIS-0546867 and DARPA award FA8750-05-2-0249. Views and conclusions contained in this document are those of the authors and do not necessarily represent the official opinion or policies, either expressed or implied, of the US government or of DARPA.

Appendix. Proof of Theorem 1

Proof. Let w^k be the weights before the k'th mistake is made. Then $w^1 = 0$. Suppose the k'th mistake is made when the beam B at depth j does not contain the target node $n^* = n^*_{ij}$. The occurrence of the mistake indicates that, for $n \in B$, $w^k \cdot \Phi(n^*) > w^k \cdot \Phi(n)$, which lets us derive an upper bound for $\|w^{k+1}\|^2$.

$$\begin{aligned}
\|w^{k+1}\|^2 &= \|w^k + \frac{\sum_{n\in B}\Phi(n)}{b} - \Phi(n^*)\|^2 \\
&= \|w^k + \sum_{n\in B}\frac{\Phi(n)-\Phi(n^*)}{b}\|^2 \\
&= \|w^k\|^2 + 2w^k \cdot \sum_{n\in B}\frac{\Phi(n)-\Phi(n^*)}{b} \\
&\quad + \|\sum_{n\in B}\frac{\Phi(n)-\Phi(n^*)}{b}\|^2 \\
&\leq \|w^k\|^2 + \|\sum_{n\in B}\frac{\Phi(n)-\Phi(n^*)}{b}\|^2 \\
&\leq \|w^k\|^2 + R^2
\end{aligned}$$

where the first equality follows from the definition of the perceptron-update rule and the first inequality follows be-

cause $w^k \cdot (\Phi(n) - \Phi(n^*)) < 0$ for each $n \in B$. Using this upper-bound we get by induction that

$$\|w^{k+1}\|^2 \leq kR^2$$

Next we derived a lower bound for $w \cdot w^{k+1}$. We will denote by $B' \subseteq B$ the set of nodes in the beam such that $\delta_1 > w \cdot (\Phi(n) - \Phi(n^*)) \geq -\delta_2$. By the definition of beam margin $|B'| \leq b'$.

$$\begin{aligned}
w \cdot w^{k+1} &= w \cdot w^k + w \cdot \sum_{n \in B} \frac{\Phi(n) - \Phi(n^*)}{b} \\
&= w \cdot w^k + w \cdot \sum_{n \in B - B'} \frac{\Phi(n) - \Phi(n^*)}{b} \\
&\quad + w \cdot \sum_{n \in B'} \frac{\Phi(n) - \Phi(n^*)}{b} \\
&\geq w \cdot w^k + w \cdot \sum_{n \in B - B'} \frac{\Phi(n) - \Phi(n^*)}{b} - \frac{b'\delta_2}{b} \\
&\geq w \cdot w^k + \frac{(b - b')\delta_1}{b} - \frac{b'\delta_2}{b}
\end{aligned}$$

By induction, we get that $w \cdot w^{k+1} \geq k\frac{(b-b')\delta_1 - b'\delta_2}{b}$. Combining this result with the above upper bound on $\|w^{k+1}\|$ and the fact that $\|w\| = 1$ we get that

$$1 \geq \frac{w \cdot w^{k+1}}{\|w\|\|w^{k+1}\|} \geq k\frac{(b - b')\delta_1 - b'\delta_2}{b\sqrt{k}R}$$

Under the theorem's assumption that $b > \left(1 + \frac{\delta_2}{\delta_1}\right) b'$ we can infer that $\delta_1(b - b') > \delta_2 b'$ and rearrange this inequality to bound the number of mistakes k by

$$k \leq \left(\frac{bR}{\delta_1(b - b') - \delta_2 b'}\right)^2$$

which completes the proof. □

References

Bonet, B., and Geffner, H. 1999. Planning as heuristic search: New results. In *ECP*, 360–372.

Boyan, J., and Moore, A. 2000. Learning evaluation functions to improve optimization by local search. *Journal of Machine Learning Research* 1:77–112.

Buro, M. 1998. From simple features to sophiscated evaluation functions. In van den Herik, H. J., and Iida, H., eds., *Proceedings of the First International Conference on Computers and Games ((CG)-98)*, volume 1558, 126–145. Tsukuba, Japan: Springer-Verlag.

Collins, M. 2002. Discriminative training methods for hidden Markov models: Theory and experiments with the perceptron algorithm. In *Conf. on Empirical Methods in NLP*.

Daume III, H., and Marcu, D. 2005. Learning as search optimization: Approximate large margin methods for structured prediction. In *International Conference on Machine Learning (ICML)*.

Fern, A.; Yoon, S.; and Givan, R. 2003. Approximate policy iteration with a policy language bias. In *Proceedings of the 16th Conference on Advances in Neural Information Processing*.

Furcy, D., and Koenig, S. 2005. Limited discrepancy beam search. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research* 14:263–302.

Huang, Y.-C.; Selman, B.; and Kautz, H. 2000. Learning declarative control rules for constraint-based planning. In *Proceedings of the 17th International Conference on Machine Learning*, 415–422. Morgan Kaufmann, San Francisco, CA.

Khardon, R. 1996. Learning to take actions. In *AAAI/IAAI, Vol. 1*, 787–792.

Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, 282–289.

Nguyen, X.; Kambhampati, S.; and Nigenda, R. S. 2002. Planning graph as the basis for deriving heuristics for plan synthesis by state space and CSP search. *Artificial Intelligence* 135(1-2):73–123.

Taskar, B.; Guestrin, C.; and Koller, D. 2003. Max-margin markov networks. In *Neural Information Processing Systems Conference*.

Tsochantaridis, I.; Hofmann, T.; Joachims, T.; and Altun, Y. 2004. Support vector machine learning for interdependent and structured output spaces. In *International Conference on Machine Learning*.

Yoon, S.; Fern, A.; and Givan, R. 2005. Learning measures of progress for planning domains. In *Proceedings of the Twentieth Conference on Uncertainty in Artificial Intelligence*.

Yoon, S.; Fern, A.; and Givan, R. 2006. Learning heuristic functions from relaxed plans. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Zhang, W., and Dietterich, T. G. 1995. A reinforcement learning approach to job-shop scheduling. In *Proceedings of the International Joint Conference on Artificial Intellience*.