Journal of Machine Learning Research 10 (2009) 1571-1610 Submitted 1/08; Revised 12/08; Published 7/09

Learning Linear Ranking Functions for Beam Search with Application to Planning

Yuehua Xu XUYU@EECS.OREGONSTATE.EDU
Alan Fern AFERN@EECS.OREGONSTATE.EDU
School of Electrical Engineering and Computer Science
Oregon State University
Kelley Engineering Center
Corvallis, OR 97330

Sungwook Yoon SUNGWOOK.YOON@PARC.COM
Palo Alto Research Center
3333 Coyote Hill Road
Palo Alto, CA 94304

Editor: Michael Littman

Abstract

Beam search is commonly used to help maintain tractability in large search spaces at the expense of completeness and optimality. Here we study supervised learning of linear ranking functions for controlling beam search. The goal is to learn ranking functions that allow for beam search to perform nearly as well as unconstrained search, and hence gain computational efficiency without seriously sacrificing optimality. In this paper, we develop theoretical aspects of this learning problem and investigate the application of this framework to learning in the context of automated planning. We first study the computational complexity of the learning problem, showing that even for exponentially large search spaces the general consistency problem is in NP. We also identify tractable and intractable subclasses of the learning problem, giving insight into the problem structure. Next, we analyze the convergence of recently proposed and modified online learning algorithms, where we introduce several notions of problem margin that imply convergence for the various algorithms. Finally, we present empirical results in automated planning, where ranking functions are learned to guide beam search in a number of benchmark planning domains. The results show that our approach is often able to outperform an existing state-of-the-art planning heuristic as well as a recent approach to learning such heuristics.

Keywords: beam search, speedup learning, automated planning, structured classification

1. Introduction

Throughout artificial intelligence and computer science, heuristic search is a fundamental approach to solving complex problems. Unfortunately, when the heuristic is not accurate enough, memory and time constraints make pure heuristic search impractical. One common way to attempt to maintain tractability of heuristic search is through a pruning technique known as beam search. At each search step, beam search maintains a "beam" of the heuristically best *b* nodes, pruning all other nodes from the search queue. Due to this pruning, beam search is not guaranteed to be complete nor optimal. However, if the heuristic is good enough to keep a good solution path in the beam, then the solution will be found quickly.

The goal of this paper is to study the problem of learning heuristics, or ranking functions, that allow beam search to quickly find solutions, without seriously sacrificing optimality compared to unconstrained search. We consider this problem for the case of linear ranking functions, where each search node v is associated with a feature vector $f(v)$ and nodes are ranked according to $w \cdot f(v)$ where w is a weight vector. Each instance in our training set corresponds to a search space that is labeled by a set of target solutions, each solution being a (satisficing) path from the initial node to a goal node. Given a training set, our learning objective is to select a weight vector w such that a beam search of a specified beam width always maintains one of the target paths in the beam until finally reaching a goal node. Such a w effectively represents a ranking function that allows beam search to efficiently solve all of the training instances, and ideally new search problems for which the training set is representative.

Recent work (Daumé III and Marcu, 2005) has considered the problem of learning beam search ranking functions in the context of structured classification. Structured classification is the problem of learning a mapping from structured inputs (e.g., sentences) to structured outputs (e.g., syntactic parses) and there has been much recent work that extends traditional classification algorithms to this setting including conditional random fields (Lafferty et al., 2001), the generalized Perceptron algorithm (Collins, 2002), and margin optimization (Taskar et al., 2003). The approach of Daumé III and Marcu (2005) differs from prior approaches in that it explicitly views structured classification as a search problem, where given an 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 is a sequence of word tags, 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 ranking function that quickly guides the search to the search node (x, y^*) where y^* is the desired output. This framework, known as *learning as search optimization (LaSO)*, has demonstrated highly competitive performance on a number of structured classification problems.

This paper builds on the LaSO framework and makes two key contributions. First, we analyze the learning problem theoretically, in terms of its computational complexity and the convergence properties of various learning algorithms. Secondly, this paper provides an empirical evaluation in the context of automated planning, a problem that is qualitatively very different from structured classification.

Our complexity analysis considers a number of subclasses of the general beam-search learning problem. First, we provide an upper bound on the complexity of the general problem by showing that even for exponentially large search spaces, which are the norm, the consistency problem (i.e., finding a w that solves all training instances) remains in NP. Next, we identify several core tractable and intractable subclasses of the beam-search learning problem. Interestingly, some of these subclasses resemble more traditional “learning to rank” problems (Agarwal and Roth, 2005) with clear analogies to applications.

Our convergence analysis studies convergence properties of perceptron-style online learning algorithms. In prior work, Daumé III and Marcu (2005) proposed a notion of linear separability for this learning problem and proved convergence of the algorithm for linearly separable data. However, here we show that result to be inaccurate for subtle reasons and give a counter example. We then propose new notions of problem margin and show that convergence can be guaranteed for revised versions of the algorithm given positive margins. For the case where training data is ambiguous, that is, where many good solutions to a search problem are not included in the target solution set, we also give sufficient conditions on the minimum beam width to guarantee convergence. This result also provides a formal characterization of the intuition that the learning problem should become easier as the beam width increases, by showing that the mistake bound decreases with increasing beam width.

While the LaSO framework has been empirically evaluated in structured classification, with impressive results, its utility in other types of search problems has not been demonstrated. Here we consider the application of a LaSO-style algorithm to automated planning, which is a problem that is qualitatively very different compared to structured classification. The planning problems we consider are most naturally viewed as goal-finding problems, where we must search for a short path to a goal node in an exponentially large graph. Rather, structured classification is most naturally viewed as an optimization problem, where we must search for a structured object that optimizes an objective function. While the two problem classes are related they differ in significant ways. For example, the search problems studied in structured classification typically have a single or small number of solution paths, whereas in automated planning there are often a large number of equally good solutions, which can contribute to ambiguous training data. Furthermore, the size of the search spaces encountered in automated planning are usually much larger than in structured classification, because of the larger depths and branching factors. These differences raise the empirical question of whether a LaSO-style approach will be effective in automated planning.

To evaluate this question we incorporated a LaSO-style learning mechanism into a forward state-space search planner in order to learn domain-specific heuristics, or ranking functions, from training examples. For a given planning domain, the training examples given to our learner include solution plans to a set of planning problems from the domain. The learned ranking function for a domain can then be used to guide beam search in order to solve new test problems from the same domain. We evaluate this approach on a number of benchmark planning domains and show that our learned ranking functions are often able to outperform both a state-of-the-art domain-independent planning heuristic and the heuristics learned by another recently proposed learning mechanism based on linear regression.

The remainder of this paper proceeds as follows. In Section 2, we introduce our formal setup of the beam-search learning problem and then, in Section 3, study the computational complexity of this learning problem. In Section 4, we describe two online learning mechanisms followed by their convergence analysis. In Section 5, we apply the learning problem to automated planning and present the experimental results. Finally Section 6 concludes and suggests future directions.

2. Problem Setup

In this section, we first describe two different beam search paradigms: breadth-first beam search and best-first beam search. We then introduce the learning problems that we study in these two paradigms, followed by an illustrative example from automated planning. Finally, we describe how our formulation, which was motivated by automated planning, relates to structured classification.

2.1 Beam Search

We first define breadth-first and best-first beam search, the two paradigms considered in this work. A *search space* is a tuple $\langle I, s(\cdot), f(\cdot), < \rangle$, where I is the initial search node, s is a successor function from search nodes to finite sets of search nodes, f is a feature function from search nodes to m-dimensional real-valued vectors, and $<$ is a total preference ordering on search nodes. We think of f as defining properties of search nodes that are useful for evaluating their relative goodness and $<$

as defining a canonical ordering on nodes, for example, lexicographic. In this work, we use f to define a linear ranking function $w \cdot f(v)$ on nodes where w is an m-dimensional weight vector, and nodes with larger values are considered to be higher ranked, or more preferred. Since a given w may assign two nodes the same rank, we use $<$ to break ties such that v is ranked higher than v' given $w \cdot f(v') = w \cdot f(v)$ and $v' < v$, arriving at a total rank ordering on search nodes. We denote this total rank ordering as $r(v', v|w, <)$, or just $r(v', v)$ when w and $<$ are clear from context, indicating that v is ranked higher than v'.

Given a search space $S = \langle I, s(\cdot), f(\cdot), < \rangle$, a weight vector w, and a beam width b, *breadth-first beam search* simply conducts breadth-first search, but at each search depth keeps only the b highest ranked nodes according to r. More formally, breadth-first beam search generates a unique *beam trajectory* $(B_0, B_1, \ldots)$ as follows,

- $B_0 = \{I\}$ is the initial beam;
- $C_{j+1} = \mathbf{BreadthExpand}(B_j, s(\cdot)) = \bigcup_{v \in B_j} s(v)$ is the depth $j+1$ *candidate set* of the depth j beam;
- B_{j+1} is the unique set of b highest ranked nodes in C_{j+1} according to the total ordering r.

Note that for any j, $|C_j| \leq cb$ and $|B_j| \leq b$, where c is the maximum number of children of any search node.

Best-first beam search is almost identical to breadth-first beam search except that we replace the function **BreadthExpand** with $\mathbf{BestExpand}(B_j, s(\cdot)) = B_j \cup s(v^*) - v^*$, where v^* is the unique highest ranking node in B_j. Thus, instead of expanding all nodes in the beam at each search step, best-first search is more conservative and only expands the single best node. Note that unlike breadth-first search this can result in beams that contain search nodes from different depths of the search space relative to I.

2.2 Learning Problems

Our learning problems provide training sets of pairs $\{\langle S_i, P_i \rangle\}$, where the $S_i = \langle I_i, s_i(\cdot), f_i(\cdot), <_i \rangle$ are search spaces constrained such that each f_i has the same dimension. As described in more detail below, the P_i encode sets of *target search paths* that describe desirable search paths through the corresponding search spaces. Roughly speaking the learning goal is to learn a ranking function that can produce a beam trajectory of a specified width for each search space that contains at least one of the corresponding target paths in the training data. For example, in the context of automated planning, the S_i would correspond to planning problems from a particular domain, encoding the state space and available actions, and the P_i would encode optimal or satisficing plans for those problems. A successfully learned ranking function would be able to quickly find at least one of the target solution plans for each training problem and ideally new target problems.

We represent each set of target search paths as a sequence $P_i = (P_{i,0}, P_{i,1}, \ldots, P_{i,d})$ of sets of search nodes where $P_{i,j}$ contains target nodes at depth j and $P_{i,0} = \{I_i\}$. It is useful to think about $P_{i,d}$ as encoding the *goal nodes* of the i'th search space. We will refer to the maximum size t of any target node set $P_{i,j}$ as the *target width* of P_i, which will be referred to in our complexity analysis. The generality of this representation for target paths allows for pathological targets where certain nodes do not lead to the goal. In order to arrive at convergence results, we rule out such possibilities by assuming that the training set is *dead-end free*. That is, for all i and $j < d$ each $v \in P_{i,j}$ has at least one child node $v' \in P_{i,j+1}$. Note that in almost all real problems this property will be naturally satisfied. For our complexity analysis, we will not need to assume any special properties of the target search paths P_i.

Intuitively, for a dead-end free training set, each P_i represents a layered directed graph with at least one path from each target node to a goal node in $P_{i,d}$. Thus, the training set specifies not only a set of goals for each search space but also gives possible solution paths to the goals. For simplicity, we assume that all target solution paths have depth d, but all results easily generalize to non-uniform depths.

For breadth-first beam search we specify a learning problem by giving a training set and a beam width $\langle \{\langle S_i, P_i \rangle\}, b \rangle$. The objective is to find a weight vector w that generates a beam trajectory containing at least one of the target paths for each training instance. More formally, we are interested in the consistency problem:

Definition 1 (Breadth-First Consistency) *Given the input $\langle \{\langle S_i, P_i \rangle\}, b \rangle$ where b is a positive integer and $P_i = (P_{i,0}, P_{i,1}, \ldots, P_{i,d})$, the breadth-first consistency problem asks us to decide whether there exists a weight vector w such that for each S_i, the corresponding beam trajectory $(B_{i,0}, B_{i,1}, \ldots, B_{i,d})$, produced using w with a beam width of b, satisfies $B_{i,j} \cap P_{i,j} \neq \emptyset$ for each j?*

A weight vector that demonstrates a "yes" answer is guaranteed to allow a breath-first beam search of width b to uncover at least one goal node (i.e., a node in $P_{i,d}$) within d beam expansions for all training instances.

Unlike the case of breadth-first beam search, the length of the beam trajectory required by best-first beam search to reach a goal node can be greater than the depth d of the target paths. This is because best-first beam search, does not necessarily increase the maximum depth of search nodes in the beam at each search step. Thus, in addition to specifying a beam width for the learning problem, we also specify a maximum number of search steps, or horizon, h. The objective is to find a weight vector that allows a best-first beam search to find a goal node within h search steps, while always keeping some node from the target paths in the beam.

Definition 2 (Best-First Consistency) *Given the input $\langle \{\langle S_i, P_i \rangle\}, b, h \rangle$, where b and h are positive integers and $P_i = (P_{i,0}, \ldots, P_{i,d})$, the best-first consistency problem asks us to decide whether there is a weight vector w that produces for each S_i a beam trajectory $(B_{i,0}, \ldots, B_{i,k})$ of beam width b, such that $k \leq h$, $B_{i,k} \cap P_{i,d} \neq \emptyset$ (i.e., $B_{i,k}$ contains a goal node), and each $B_{i,j}$ for $j < k$ contains at least one node in $\bigcup_j P_{i,j}$?*

Again, a weight vector that demonstrates a "yes" answer is guaranteed to allow a best-first beam search of width b to find a goal node in h search steps for all training instances.

2.2.1 EXAMPLE FROM AUTOMATED PLANNING.

Figure 1, shows a pictorial example of a single training example from an automated planning problem. The planning domain in this example is Blocksworld where individual problems involve transforming an initial configuration of blocks to a goal configuration using simple actions such as picking up, putting down, and stacking the various blocks. The figure shows a search space S_i where each node corresponds to a configuration of blocks and the arcs indicate when it is possible to take an action that transitions from one configuration to another. The figure depicts, via highlighted nodes, two target paths. The label P_i would encode these target paths by a sequence $P_i = (P_{i,0}, P_{i,1}, \ldots, P_{i,4})$ where $P_{i,j}$ contains the set of all highlighted target nodes at depth j. A solution weight vector, for this training example, would be required to keep at least one of the highlighted paths in the beam until uncovering the goal node.

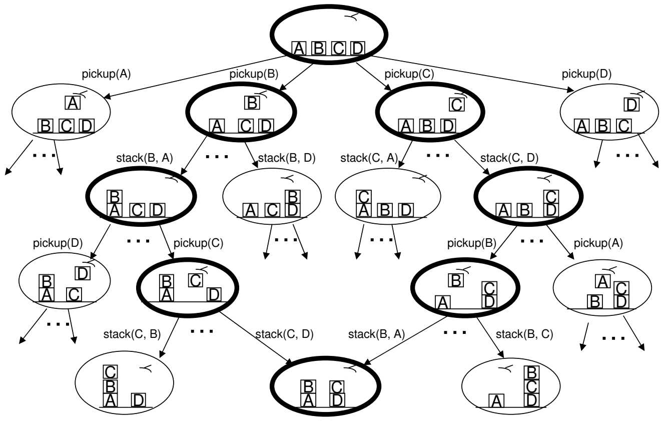

Figure 1: An example from automated planning.

2.2.2 EXAMPLE FROM STRUCTURED CLASSIFICATION

Daumé III and Marcu (2005) considered learning ranking functions to control beam search in the context of structured classification. Structured classification involves learning a function that maps structured inputs x to structured outputs y. As an example, consider part-of-speech tagging where the inputs correspond to English sentences and the correct output for a sentence is the sequence of part-of-speech tags for the words in the sentence. Figure 2 shows how Daumé III and Marcu (2005) formulated a single instance of part-of-speech tagging as a search problem. Each search node is a pair (x, y') where x is the input sentence and y' is a partial labeling of the words in x by part-of-speech tags. The arcs in this space correspond to search steps that label words in the sentence in a left-to-right order by extending y' in all possible ways by one element. The leaves, or terminal nodes, of this space correspond to all possible complete labelings of x. Given a ranking function and a beam width, Daumé III and Marcu (2005) return a predicted output for x by conducting a beam search until a terminal node becomes the highest ranked node in the beam, and then return the output component of that terminal node. This approach to making predictions suggests that the learning objective should require that we learn a ranking function such that the goal terminal node, is the first terminal node to become highest ranked in the beam. In the figure, there is a single goal terminal node (x, y) where y is the correct labeling of x and there is a unique target path to this goal.

From the above example, we see that there is a difference between the learning objective used by Daumé III and Marcu (2005) for structured classification and the learning objective under our formulation, which was motivated by automated planning. In particular, our formulation does not force the goal node to be the highest ranked node in the final beam, but rather only requires that a goal node appear somewhere in the final beam. While these formulations appear quite different, it turns out that they are polynomially reducible to one another, which we prove in Appendix A.

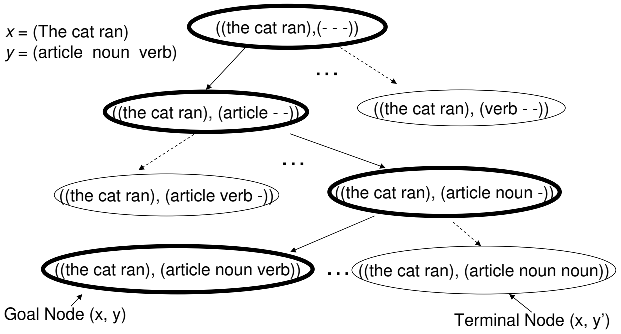
x = (The cat ran)
y = (article noun verb)
((the cat ran),(- - -))
...
((the cat ran), (article - -))
((the cat ran), (verb - -))
...
((the cat ran), (article verb -))
((the cat ran), (article noun -))
((the cat ran), (article noun verb))
...
((the cat ran), (article noun noun))
Goal Node (x, y)
Terminal Node (x, y')

Figure 2: An example from structured classification.

Thus, all of the results in this paper apply equally well to the structured-classification formulation of Daumé III and Marcu (2005).

3. Computational Complexity

In this section, we study the computational complexity of the above consistency problems. We first focus on breadth-first beam search, and then give the corresponding best-first results at the end of this section. It is important to note that the size of the search spaces will typically be exponential in the encoding size of the learning problem. For example, in automated planning, standard languages such as PDDL (McDermott, 1998) are used to compactly encode planning problems that are potentially exponentially large, in terms of the number of states, with respect to the PDDL encoding size. Throughout this section we measure complexity in terms of the problem encoding size, not the potentially exponentially larger search space size. All discussions in this section apply to general search spaces and are not tied to a particular language for describing search space such as PDDL.

Our complexity analysis will consider various sub-classes of the breadth-first consistency problem, where the sub-classes will be defined by placing constraints on the following problem parameters: n - the number of training instances, d - the depth of target solution paths, c - the maximum number of children of any search node, t - the maximum target width of any P_i as defined in Section 2.2, and b - the beam width. Figure 3 gives a pictorial depiction of these key problem parameters. Throughout the complexity analysis we will restrict our attention to problem classes where the maximum number of children c and beam width b are polynomial in the problem size, which are necessary conditions to ensure that each beam search step requires only polynomial time and space. We will also assume that all feature functions can be evaluated in polynomial time in the problem size.

Note that restricting the number of children c may rule out the use of certain search space encodings for some problems. For example, in a multi-agent planning scenario, there are an exponential number of joint actions to consider from each state, and thus an exponential number of children. However, here it is possible to re-encode the search space by increasing the depth of the search tree, so that each joint action is encoded by a sequence of steps where each agent selects an action in turn followed by all of them executing the selected actions. The resulting search space has only a polynomial number of children and thus satisfies our assumption, though the required search depth has increased. This form of re-encoding from a search space with exponentially many children to one with polynomially many children can be done whenever the actions in the original space have a compact, factored encoding, which is typically the case in practice.

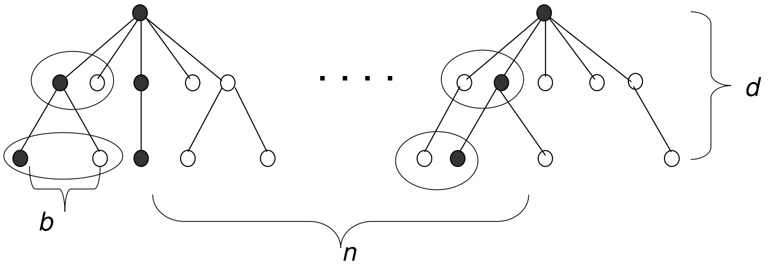

Figure 3: The key problem parameters: n - the number of training instances, d - the depth of target solution paths, b - the beam width. Not depicted in the figure are: c - maximum number of children of any node, t - the maximum target width of any example.

3.1 Hardness Upper Bounds

We first show an upper bound on the complexity of breadth-first consistency by proving that the general problem is in NP even for exponentially large search spaces.

Observe that given a weight vector w and beam width b, we can easily generate a unique depth d beam trajectory for each training instance. Our upper bound is based on considering the inverse problem of checking whether a set of hypothesized beam trajectories, one for each training instance, could have been generated by some weight vector. The algorithm *TestTrajectories* in Figure 4 efficiently carries out this check. The main idea is to observe that for any search space S it is possible to efficiently check whether there is a weight vector that starting with a beam B could generate a beam B' after one step of breadth-first beam search. This can be done by constructing an appropriate set of linear constraints on the weight vector w that are required to generate B' from B. In particular, we first generate the set of candidate nodes C from B by unioning all children of nodes in B. Clearly we must have $B' \subseteq C$ in order for there to be a solution weight vector. If this is the case then we create a linear constraint for each pair of nodes (u,v) such that $u \in B'$ and $v \in C - B'$, which forces u to be preferred to v:

$$w \cdot f(u) > w \cdot f(v)$$

where $w = (w_1, w_2, \ldots, w_m)$ are the constraint variables and $f(\cdot) = (f_1(\cdot), f_2(\cdot), \ldots, f_m(\cdot))$ is the vector of feature functions. Note that if u is more preferred than v in the total preference ordering, then we only need to require that $w \cdot f(u) \geq w \cdot f(v)$. The overall algorithm *TestTrajectories* simply creates this set of constraints for each consecutive pair of beams in each beam trajectory and then tests to see whether there is a w that satisfies all of the constraints.

Lemma 3 *Given a set of search spaces $\{S_i\}$ and a corresponding set of width b beam trajectories $\{(B_{i,0}, \ldots, B_{i,d})\}$, the algorithm TestTrajectories (Figure 4) decides in polynomial time whether there exists a weight vector w that can generate $(B_{i,0}, \ldots, B_{i,d})$ in S_i for all i.*

Proof It is straightforward to show that w satisfies the constraints generated by *TestTrajectories* iff for each i, j, $r(v', v| <_i, w)$ leads beam search to generate $B_{i,j+1}$ from $B_{i,j}$. The linear program contains m variables and at most $ndcb^2$ constraints. Since we are assuming that the maximum number of children of a node v is polynomial in the size of the learning problem, the size of the linear program is also polynomial and thus can be solved in polynomial time (Khachiyan, 1979). ■

This lemma shows that sets of beam trajectories can be used as efficiently-checkable certificates for breadth-first consistency, which leads to an upper bound on the problem's complexity.

Theorem 4 *Breadth-first consistency is in NP.*

Proof Given a learning problem $\langle\{\langle S_i, P_i\rangle\}, b\rangle$ our certificates correspond to sets of beam trajectories $\{(B_{i,0}, \ldots, B_{i,d})\}$ each of size at most $O(ndb)$ which is polynomial in the problem size. The certificate can then be checked in polynomial time to see if for each i, $(B_{i,0}, \ldots, B_{i,d})$ contains a target solution path encoded in P_i as required by Definition 1. If it is consistent then according to Lemma 3 we can efficiently decide whether there is a w that can generate $\{(B_{i,0}, \ldots, B_{i,d})\}$. ■

This result suggests an enumeration-based decision procedure for breadth-first consistency as given in Figure 4. In that procedure, the function **Enumerate** creates a list of all possible combinations of beam trajectories for the training data. Thus, each element of this list is a list of beam trajectories, one for each training example, where a beam trajectory is simply a sequence of sets of nodes that are selected from the given search space. For each enumerated combination of beam trajectories, the function **IsConsistent** checks whether the beam trajectory for each example contains a target path for that example and if so *TestTrajectories* will be called to determine whether there exists a weight vector that could produce those trajectories. The following gives us the worst case complexity of this algorithm in terms of the key problem parameters.

Theorem 5 *The procedure ExhaustiveAlgorithm (Figure 4) decides breadth-first consistency and returns a solution weight vector if there is a solution in time* $O\left((t + poly(m))\,(cb)^{bdn}\right)$.

Proof We first bound the number of certificates. Breadth-first beam search expands nodes in the current beam, resulting in at most cb nodes, from which b nodes are selected for the next beam. Enumerating these possible choices over d levels and n trajectories, one for each training instance, we can bound the number of certificates by $O\left((cb)^{bdn}\right)$. For each certificate the enumeration process checks consistency with the target paths $\{P_i\}$ in time $O(tbdn)$ and then calls *TestTrajectories* which runs in time $\mathrm{poly}(m, ndcb^2)$. The total time complexity then is $O\left(\left(tbdn + \mathrm{poly}(m, ndcb^2)\right)(cb)^{bdn}\right)$ $= O\left((t + \mathrm{poly}(m))\,(cb)^{bdn}\right)$. ■

Not surprisingly the complexity is exponential in the beam width b, target path depth d, and number of training instances n. However, it is polynomial in the maximum number of children c and the maximum target width t. Thus, breadth-first consistency can be solved in polynomial time for any problem class where b, d, and n are constants. Of course, for most problems these constants would be too large for this result to be of practical interest. This leads to the question of whether we can do better than the exhaustive algorithm for restricted problem classes. For at least one problem class we can.

```
ExhaustiveAlgorithm ($\{\langle S_i, P_i \rangle\}, b$)
Γ = Enumerate($\{\langle S_i, P_i \rangle\}, b$)
// enumerates all possible sets of beam trajectories
for each $\{(B_{i,0} \ldots, B_{i,d})\} \in \Gamma$
    if IsConsistent($\{P_i\}, \{(B_{i,0} \ldots, B_{i,d})\}$) then
        $w$= TestTrajectories($\{S_i\}, \{(B_{i,0}, \ldots, B_{i,d})\}$)
        if $w \neq$ false then
            return $w$
return false
```

```
TestTrajectories($\{S_i\}, \{(B_{i,0}, \ldots, B_{i,d})\}$)
// $S_i = \langle I_i, s_i(\cdot), f_i(\cdot), <_i \rangle$
construct a linear programming problem $LP$ as below
    the variables are $w = \{w_1, w_2, \ldots, w_m\}$
    for $(i, j) \in \{1, \ldots, n\} \times \{1, \ldots, d\}$
        $C_{i,j}$ =BreadthExpand($B_{i,j-1}, s_i(\cdot)$)
        if $B_{i,j} \subseteq C_{i,j}$ then
            for each $u \in B_{i,j}$ and $v \in C_{i,j} - B_{i,j}$
                if $v <_i u$ then
                    add a constraint $w \cdot f_i(u) \geq w \cdot f_i(v)$
                else add a constraint $w \cdot f_i(u) > w \cdot f_i(v)$
        else return false
$w$ = LPSolver($LP$)
if $LP$ is solved then
    return $w$
return false
```

Figure 4: The exhaustive algorithm for breadth-first consistency.

Theorem 6 *The class of breadth-first consistency problems where $b = 1$ and $t = 1$ is solvable in polynomial time.*

Proof Given a learning problem $\langle \{\langle S_i, P_i \rangle\}, b \rangle$ where $P_i = (P_{i,0}, \ldots, P_{i,d})$, $t = 1$ implies that each $P_{i,j}$ contains exactly one node. Since the beam width $b = 1$, then the only way that a beam trajectory $(B_{i,0}, \ldots, B_{i,d})$ can satisfy the condition $B_{i,j} \cap P_{i,j} \neq \emptyset$ for any i, j, is for $B_{i,j} = P_{i,j}$. Thus there is exactly one beam trajectory for each training example, equal to the target trajectory, and using Lemma 3 we can check for a solution weight vector for these trajectories in polynomial time. ■

This problem class, as depicted in Figure 5, corresponds to the case where each training instance is labeled by exactly a single solution path and we are asked to find a w that leads a greedy hill-climbing search, or reactive policy, to follow those paths. This is a common learning setting, for example, when attempting to learn reactive control policies based on demonstrations of target policies, perhaps from an expert, as in Khardon (1999).

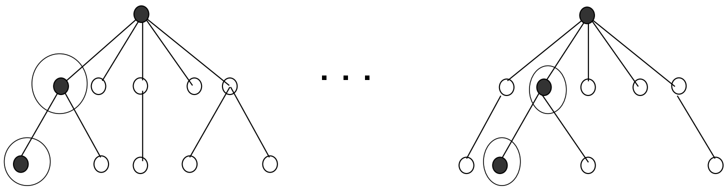

Figure 5: A tractable class of breadth-first consistency, where $b = 1$ and $t = 1$.

3.2 Hardness Lower Bounds

Unfortunately, outside of the above problem classes it appears that breadth-first consistency is computationally hard even under strict restrictions. In particular, the following three results show that if any one of b, d, or n are not bounded then the consistency problem is hard even when the other problem parameters are small constants.

First, we show that the problem class where $n = d = t = 1$ but $b \geq 1$ is NP-complete. That is, a single training instance involving a depth one search space is sufficient for hardness. This problem class, resembles more traditional ranking problems and has a nice analogy in the application domain of web-page ranking, where the depth 1 leaves of our search space correspond to possibly relevant web-pages for a particular query. One of those pages is marked as a target page, for example, the page that a user eventually went to. The learning problem is then to find a weight vector that will cause for the target page to be ranked among the top b pages. Our result shows that this problem is NP-complete and hence will be exponential in b unless $P = NP$.

Theorem 7 *The class of breadth-first consistency problems where $n = 1$, $d = 1$, $t = 1$, and $b \geq 1$ is NP-complete.*

Proof Our reduction is from the Minimum Disagreement problem for linear binary classifiers, which was proven to be NP-complete by Hoffgen et al. (1995). The input to this problem is a training set $T = \{x_1^+, \cdots, x_{r_1}^+, x_1^-, \cdots, x_{r_2}^-\}$ of positive and negative m-dimensional vectors and a positive integer k. A weight vector w classifies a vector as positive iff $w \cdot x \geq 0$ and otherwise as negative. The Minimum Disagreement problem is to decide whether there exists a weight vector that commits no more than k misclassification.

Given a Minimum Disagreement problem we construct an instance $\langle\langle S_1, P_1\rangle, b\rangle$ of the breadth-first consistency problem as follows. Assume without loss of generality $S_1 = \langle I, s(\cdot), f(\cdot), <\rangle$. Let $s(I) = \{q_0, q_1, \cdots, q_{r_1+r_2}\}$. For each $i \in \{1, \cdots, r_1\}$, define $f(q_i) = -x_i^+ \in R^m$. For each $i \in \{1, \cdots, r_2\}$,define $f(q_{i+r_1}) = x_i^- \in R^m$. Define $f(q_0) = 0 \in R^m$, $P_1 = (\{I\}, \{q_0\})$ and $b = k+1$. Define the total ordering $<$ to be a total ordering in which $q_i < q_0$ for every $i = 1, \ldots, r_1$ and $q_0 < q_i$ for every $i = r_1+1, \ldots, r_1+r_2$.We claim that there exists a linear classifier with at most k misclassifications if and only if there exists a solution to the corresponding consistency problem.

First, suppose there exists a linear classifier $w \cdot x \geq 0$ with at most k misclassifications. Using the weight vector w, we have

- $w \cdot f(q_0) = 0$;

- for $i = 1, \cdots, r_1$:
  if $w \cdot x_i^+ \geq 0$, $w \cdot f(q_i) = w \cdot (-x_i^+) \leq 0$;
  if $w \cdot x_i^+ < 0$, $w \cdot f(q_i) = w \cdot (-x_i^+) > 0$;

- for $i = r_1 + 1, \ldots, r_1 + r_2$:
  if $w \cdot x_i^- \geq 0$, $w \cdot f(q_i) = w \cdot x_i^- \geq 0$;
  if $w \cdot x_i^- < 0$, $w \cdot f(q_i) = w \cdot x_i^- < 0$.

For $i = 1, \cdots, r_1 + r_2$, the node q_i in the consistency problem is ranked lower than q_0 if and only if its corresponding example in the Minimum Disagreement problem is labeled correctly, is ranked higher than q_0 if and only if its corresponding example in the Minimum Disagreement problem is labeled incorrectly. Therefore, there are at most k nodes which are ranked higher than q_0. With beam width $b = k + 1$, the beam $B_{i,1}$ is guaranteed to contain node q_0, indicating that w is a solution to the consistency problem.

On the other hand, suppose there exists a solution w to the consistency problem. There are at most $b - 1 = k$ nodes that are ranked higher than q_0. That is, at least $r_1 + r_2 - k$ nodes are ranked lower than q_0. For $i = 1, \ldots, r_1$, q_i is ranked lower than q_0 if and only if $w \cdot f(q_i) \leq w \cdot f(q_0)$. For $i = r_1 + 1, \ldots, r_1 + r_2$, q_i is ranked lower than q_0 if and only if $w \cdot f(q_i) < w \cdot f(q_0)$. Since $w \cdot f(q_0) = 0$, we have

- for $i = 1, \cdots, r_1$:
  $w \cdot f(q_i) \leq 0 \Rightarrow w \cdot (-x_i^+) \leq 0 \Rightarrow w \cdot x_i^+ \geq 0$;

- for $i = r_1 + 1, \ldots, r_1 + r_2$:
  $w \cdot f(q_i) < 0 \Rightarrow w \cdot x_i^- < 0 \Rightarrow w \cdot x_i^- < 0$.

Therefore, using the linear classifier $w \cdot x \geq 0$, at least $r_1 + r_2 - k$ nodes are labeled correctly, that is, it makes at most k misclassifications.

Since the time required to construct the instance $\langle\langle S_1, P_1\rangle, b\rangle$ from T, k is polynomial in the size of T, k, we conclude that the consistency problem is NP-Complete even restricted to $n = 1$, $d = 1$ and $t = 1$. ■

The next result shows that if we do not bound the number of training instances n, then the problem remains hard even when the target path depth and beam width are equal to one. Interestingly, this subclass of breadth-first consistency corresponds to the multi-label learning problem as defined in Jin and Ghahramani (2002). In multi-label learning each training instance can be viewed as a bag of m-dimensional vectors, some of which are labeled as positive, which in our context correspond to the target nodes. The learning goal is to find a w that for each bag, ranks one of the positive vectors as best.

Theorem 8 *The class of breadth-first consistency problems where $d = 1$, $b = 1$, $c = 6$, $t = 3$, and $n \geq 1$ is NP-complete.*

Proof The proof is by reduction from 3-SAT (Garey and Johnson, 1979), which is the following.

"Given a set U of boolean variables, a collection Q of clauses over U such that each clause $q \in Q$ has $|q| = 3$, decide whether there a satisfying truth assignment for C."

Let $U = \{u_1,\ldots,u_m\}$, $Q = \{q_{11} \vee q_{12} \vee q_{13},\ldots,q_{n1} \vee q_{n2} \vee q_{n3}\}$ be an instance of the 3-SAT problem. Here, $q_{ij} = u$ or $\neg u$ for some $u \in U$. We construct from U,Q an instance $\langle\{\langle S_i,P_i\rangle\},b\rangle$ of the breadth-first consistency problem as follows. For each clause $q_{i1} \vee q_{i2} \vee q_{i3}$, let $s_i(I_i) = \{p_{i,1},\cdots,p_{i,6}\}$, $P_i = (\{I_i\},\{p_{i,1},p_{i,2},p_{i,3}\})$, $b = 1$, and the total ordering $<_i$ is defined so that $p_{i,j} <_i p_{i,k}$ for $j = 1,2,3$ and $k = 4,5,6$. Let $e_k \in \{0,1\}^m$ denote a vector of zeros except a 1 in the k'th component. For each $i = 1,\ldots,n$, $j = 1,2,3$, if $q_{ij} = u_k$ for some k then $f_i(p_{i,j}) = e_k$ and $f_i(p_{i,j+3}) = -e_k/2$, otherwise if $q_{ij} = \neg u_k$ for some k then $f_i(p_{i,j}) = -e_k$ and $f_i(p_{i,j+3}) = e_k/2$. We claim that there exists a satisfying truth assignment if and only if there exists a solution to the corresponding consistency problem.

First, suppose that there exists a satisfying truth assignment. Let $w = (w_1,\cdots,w_m)$, where $w_k = 1$ if u_k is true, and $w_k = -1$ if u_k is false in the truth assignment. For each $i = 1,\ldots,n$, $j = 1,\ldots,3$, we have:

- if q_{ij} is true, then
  $w \cdot f_i(p_{i,j}) = 1$ and $w \cdot f_i(p_{i,j+3}) = -1/2$;

- if q_{ij} is false, then
  $w \cdot f_i(p_{i,j}) = -1$ and $w \cdot f_i(p_{i,j+3}) = 1/2$.

Note that for each clause $q_{i1} \vee q_{i2} \vee q_{i3}$, at least one of the literals is true. Thus, for every set of nodes $\{p_{i,1},p_{i,2},p_{i,3}\}$, at least one of the nodes will have the highest rank value equal to 1, resulting in $B_{i,1} = \{v\}$ where $v \in \{p_{i,1},p_{i,2},p_{i,3}\}$. By the definition, the weight vector w is a solution to the consistency problem.

On the other hand, suppose that there exists a solution $w = (w_1,\ldots,w_m)$ to the consistency problem. Assume the beam trajectory for each i is $(\{I_i\},\{v_i\})$. Then $v_i = p_{i,j}$ for some $j \in \{1,2,3\}$, and for this i and j, $q_{ij} = u_k$ or $\neg u_k$ for some k. Let u_k be true if $q_{ij} = u_k$ and be false if $q_{ij} = \neg u_k$. As long as there is no contradiction in this assignment, this is a satisfying truth assignment because at least one of $\{q_{i1},q_{i2},q_{i3}\}$ is true for every i, that is, every clause is true.

Now we will prove that there is no contradiction in this assignment, that is, any variable is assigned either true or false, but not both. Note that for any node $v \in \{p_{i,1},p_{i,2},p_{i,3}\}$, there always exists a node $v' \in \{p_{i,4},\ldots,p_{i,6}\}$ such that:

- $w \cdot f_i(v) < 0 \Leftrightarrow w \cdot f_i(v') > 0$;

- $w \cdot f_i(v) > 0 \Leftrightarrow w \cdot f_i(v') < 0$;

- $w \cdot f_i(v) = 0 \Leftrightarrow w \cdot f_i(v') = 0$.

Then because of the total ordering $<_i$ we defined, the node $v_i = p_{i,j}$ appearing in the beam trajectory, must has $w \cdot f_i(v_i) > 0$. Assume without loss of generality that $q_{ij} = u_k$, then u_k is assigned to be true. Although $\neg u_k$ might appear in other clauses, for example, $q_{i'j'} = \neg u_k$, its corresponding node $p_{i',j'}$ can never appear in the beam trajectory because $w \cdot f_{i'}(p_{i',j'}) = w \cdot (-e_k) = -w \cdot e_k = -w \cdot f_i(p_{i,j}) < 0$. Therefore, u_k will never be assigned false. A similar proof can be given for the case of $q_{ij} = \neg u_k$.

Since the time required to construct the instance $\langle\{\langle S_i,P_i\rangle\},b\rangle$ from U,Q is polynomial in the size of U,Q, we conclude that the consistency problem is NP-Complete for the case of $d = 1$, $b = 1$, $c = 6$ and $t = 3$.

■

Finally, we show that when the depth d is unbounded the consistency problem remains hard even when $b = n = 1$.

Theorem 9 *The class of breadth-first consistency problems where $n = 1$, $b = 1$, $c = 6$, $t = 3$, and $d \geq 1$ is NP-complete.*

Proof Assume $x = \langle \{\langle S_i, P_i\rangle | i = 1, \ldots, n\}, b\rangle$, where $S_i = \langle I_i, s_i(\cdot), f_i(\cdot), <_i\rangle$ and $P_i = (\{I_i\}, P_{i,1})$, is an instance of the consistency problem with $d = 1$, $b = 1$, $c = 6$ and $t = 3$. We can construct an instance y of the consistency problem with $n = 1$, $b = 1$, $c = 6$, and $t = 3$. Let $y = \langle\langle \overline{S_1}, \overline{P_1}\rangle, b\rangle$ where $\overline{S_1} = \langle I_1, \overline{s}(\cdot), \overline{f}(\cdot), \overline{<}\rangle$, and $\overline{P_1} = (\{I_1\}, P_{1,1}, P_{2,1}, \ldots, P_{t,1})$. We define $\overline{s}(\cdot)$, $\overline{f}(\cdot)$, $\overline{<}$ as below.

- $\overline{s}(I_1) = s_1(I_1)$, $\overline{f}(I_1) = f_1(I_1)$;
- for each $i = 1, \ldots, n-1$
  $\forall v \in s_i(I_i)$, $\overline{f}(v) = f_i(v)$ and $\overline{s}(v) = s_{i+1}(I_{i+1})$;
  $\forall (v, v') \in s_i(I_i)$, $\overline{<}(v, v') =<_i (v, v')$;
- $\forall v \in s_n(I_n)$, $\overline{f}(v) = f_n(v)$;
  $\forall (v, v') \in s_n(I_n)$, $\overline{<}(v, v') =<_n (v, v')$.

Obviously, a weight vector w is a solution for the instance x if and only if w is a solution for the constructed instance y. ■

b	n	d	c	t	Complexity
poly	≥ 1	≥ 1	poly	≥ 1	NP
K	K	K	poly	≥ 1	P
1	≥ 1	≥ 1	poly	1	P
poly	1	1	poly	1	NP-Complete
1	≥ 1	1	6	3	NP-Complete
1	1	≥ 1	6	3	NP-Complete

Figure 6: Complexity results for breadth-first consistency. Each row corresponds to a sub-class of the problem and indicates the computational complexity. K indicates a constant value and "poly" indicates that the quantity must be polynomially related to the problem size.

Figure 6 summarizes our main complexity results from this section for breadth-first consistency. For best-first beam search, most of these results can be carried over. Recall that for best-first consistency the problem specifies a search horizon h in addition to a beam width. Using a similar approach as above we can show that best-first consistency is in NP assuming that h is polynomial in the problem size, which is a reasonable assumption. Similarly, one can extend the polynomial time result for fixed b, n, and d. The remaining results in the table can be directly transferred to best-first search, since in each case either $b = 1$ or $d = 1$ and best-first beam search is equivalent to breadth-first beam search in either of these cases.

4. Convergence of Online Updates

In the previous section, we identified a limited set of tractable problem classes and saw that even very restricted classes remain NP-hard. We also saw that some of these hard classes had interesting application relevance. Thus, it is desirable to consider efficient learning mechanisms that work well in practice. Below we describe two such algorithms based on online perceptron updates.

4.1 Online Perceptron Updates

Figure 7 gives the LaSO-BR algorithm for learning ranking functions for breadth-first beam search. It resembles the *learning as search optimization (LaSO)* algorithm for best-first search by Daumé III and Marcu (2005). LaSO-BR iterates through all training instances $\langle S_i, P_i \rangle$ and for each one conducts a beam search of the specified width. After generating the depth j beam for the ith training instance, if at least one of the target nodes in $P_{i,j}$ are in the beam then no weight update occurs. Rather, if none of the target nodes in $P_{i,j}$ are in the beam then a search error is flagged and weights are updated according to the following perceptron-style rule,

$$w = w + \alpha \cdot \left(\frac{\sum_{v^* \in P_{i,j} \cap C} f(v^*)}{|P_{i,j} \cap C|} - \frac{\sum_{v \in B} f(v)}{b} \right)$$

where $0 < \alpha \leq 1$ is a learning rate parameter, B is the current beam and C is the candidate set from which B was generated (i.e., the beam expansion of the previous beam). For simplicity of notation, here we assume that f is a feature function for all training instances. Intuitively this weight update moves the weights in the direction of the average feature function of target nodes that appear in C, and away from the average feature function of non-target nodes in the beam. This has the effect of increasing the rank of target nodes in C and decreasing the rank of non-targets in the beam. Ideally, this will cause at least one of the target nodes to become preferred enough to remain on the beam next time through the search. Note that the use of averages over target and non-target nodes is important so as to account for the different sizes of these sets of nodes. After each weight update, the beam is reset to contain only the set of target nodes in C and the beam search then continues. Importantly, on each iteration, the processing of each training instance is guaranteed to terminate in d search steps.

Figure 8 gives the LaSO-BST algorithm for learning in best-first beam search, which is a slight modification of the original LaSO algorithm. The main difference compared to the original LaSO is in the weight update equation, a change that appears necessary for our convergence analysis. The algorithm is similar to LaSO-BR except that a best-first beam search is conducted, which means that termination for each training instance is not guaranteed to be within d steps. Rather, the number of search steps for a single training instance remains unbounded without further assumptions, which we will address later in this section. In particular, there is no bound on the number of search steps between weight updates for a given training example. This difference between LaSO-BR and LaSO-BST was of great practical importance in our automated planning application. In particular, LaSO-BST typically did not produce useful learning results due to the fact that the number of search steps between weight updates was extremely large. Note that in the case of structured classification, Daumé III and Marcu (2005) did not experience this difficulty due to the bounded-depth nature of their search spaces.

```
**LaSO-BR** ($\{\langle S_i, P_i \rangle\}, b$)
$w \leftarrow 0$
**repeat** until $w$ is unchanged **or** a large number of iterations
    **for** every $i$
        **Update-BR**($S_i, P_i, b, w$)
**return** $w$

**Update-BR** ($S_i, P_i, b, w$)
// $S_i = \langle I_i, s_i(\cdot), f(\cdot), <_i \rangle$ and $P_i = (P_{i,0}, \ldots, P_{i,d})$
$B \leftarrow \{I_i\}$ // initial beam
**for** $j = 1, \ldots, d$
    $C \leftarrow$ **BreadthExpand**($B, s_i(\cdot)$)
    **for** every $v \in C$
        $H(v) \leftarrow w \cdot f(v)$ // compute heuristic value of $v$
    Order $C$ according to $H$ and the total ordering $<_i$
    $B \leftarrow$ the first $b$ nodes in $C$
    **if** $B \cap P_{i,j} = \emptyset$ **then**
        $w \leftarrow w + \alpha \cdot \left( \frac{\sum_{v^* \in P_{i,j} \cap C} f(v^*)}{|P_{i,j} \cap C|} - \frac{\sum_{v \in B} f(v)}{b} \right)$
        $B \leftarrow P_{i,j} \cap C$
**return**
```

Figure 7: The LaSO-BR online algorithm for breadth-first beam search.

4.2 Previous Result and Counter Example

Adjusting to our terminology, Daumé III and Marcu (2005) defined a training set to be *linear separable* iff there is a weight vector that solves the corresponding consistency problem. Also for linearly separable data they defined a notion of margin of a weight vector, which we refer to here as the *search margin*. The formal definition of search margin is given below.

Definition 10 (Search Margin) *The search margin of a weight vector w for a linearly separable training set is defined as $\gamma = min_{\{(v^*,v)\}}(w \cdot f(v^*) - w \cdot f(v))$, where the set $\{(v^*,v)\}$ contains any pair where v^* is a target node and v is a non-target node that was compared during the beam search guided by w.*

Daumé III and Marcu (2005) state that the existence of a w with positive search margin, which implies linear separability, implies convergence of the original LaSO algorithm after a finite number of weight updates. On further investigation, we have found that a positive search margin is not sufficient to guarantee convergence for LaSO, LaSO-BR, or LaSO-BST. Intuitively, the key difficulty is that our learning problem contains hidden state in the form of the desired beam trajectory. Given the beam trajectory of a consistent weight vector one can compute the weights, and likewise given consistent weights one can compute the beam trajectory. However, we are given neither to begin with and our approach can be viewed as an online EM-style algorithm, which alternates between

```
LaSO-BST ($\{\langle S_i, P_i \rangle\}, b$)
$w \leftarrow 0$
repeat until $w$ is unchanged or a large number of iterations
    for every $i$
        Update-BST($S_i, P_i, b, w$)
return $w$
```

```
Update-BST ($S_i, P_i, b, w$)
// $S_i = \langle I_i, s_i(\cdot), f(\cdot), <_i \rangle$ and $P_i = (P_{i,0}, \ldots, P_{i,d})$
$B \leftarrow \{I_i\}$ // initial beam
$\bar{P} = P_{i,0} \cup P_{i,2} \cup \ldots \cup P_{i,d}$
while $B \cap P_{i,d} = \emptyset$
    $C \leftarrow$ BestExpand($B, s_i(\cdot)$)
    for every $v \in C$
        $H(v) \leftarrow w \cdot f(v)$ // compute heuristic value of $v$
    Order $C$ according to $H$ and the total ordering $<_i$
    $B \leftarrow$ the first $b$ nodes in $C$
    if $B \cap \bar{P} = \emptyset$ then
        $w \leftarrow w + \alpha \cdot \left( \frac{\sum_{v^* \in \bar{P} \cap C} f(v^*)}{|P \cap C|} - \frac{\sum_{v \in B} f(v)}{b} \right)$
        $B \leftarrow \bar{P} \cap C$
return
```

Figure 8: Online algorithm for best-first beam search.

updating weights given the current beam and recomputing the beam given the updated weights. Just as traditional EM is quite prone to local minima, so are the LaSO algorithms in general, and in particular even when there is a positive search margin as demonstrated in the following counter example. Note that the standard Perceptron algorithm for classification learning does not run into this problem since there is no hidden state involved.

Counter Example 1 *We give a training set for which the existence of a weight vector with positive search margin does not guarantee convergence to a solution weight vector for LaSO-BR or LaSO-BST. Consider a problem that consists of a single training instance with search space shown in Figure 9, preference ordering $C < B < F < E < D < H < G$, and single target path $P = (\{A\}, \{B\}, \{E\})$.*

First we will consider using breadth-first beam search with a beam width of $b = 2$. Using the weight vector $w = [\gamma, \gamma]$ the resulting beam trajectory will be (note that higher values of $w \cdot f(v)$ are better):

$$\{A\}, \{B,C\}, \{E,F\}.$$

The search margin of w, which is only sensitive to pairs of target and non-target nodes that were compared during the search, is equal to,

$$\gamma = w \cdot f(B) - w \cdot f(C) = w \cdot f(E) - w \cdot f(F)$$

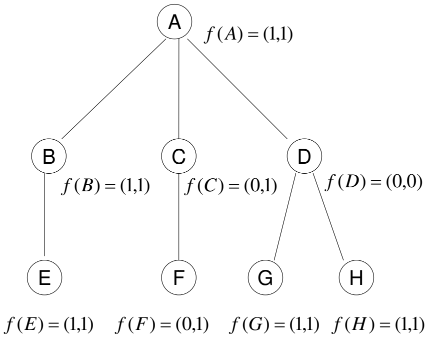

Figure 9: Counter example for convergence under positive search margin.

which is positive. We now show that the existence of w does not imply convergence under perceptron updates.

Consider simulating LaSO-BR starting from $w' = 0$. The first search step gives the beam $\{D,B\}$ according to the given preference ordering. Since B is on the target path we continue expanding to the next level where we get the new beam $\{G,H\}$. None of the nodes are on the target path so we update the weights as follows:

$$\begin{aligned} w' &= w' + f(E) - 0.5[f(G) + f(H)] \\ &= w' + [1,1] - [1,1] \\ &= w'. \end{aligned}$$

This shows that w' does not change and we have converged to the weight vector $w' = 0$, which is not a solution to the problem.

For the case of best-first beam search, the performance is similar. Given the weight vector $w = [\gamma, \gamma]$, the resulting beam search with beam width 2 will generate the beam sequence,

$$\{A\}, \{B,C\}, \{E,C\}$$

which is consistent with the target trajectory. From this we can see that w has a positive search margin of:

$$\gamma = w \cdot f(B) - w \cdot f(C) = w \cdot f(E) - w \cdot f(C).$$

However, if we follow the perceptron algorithm when started with the weight vector $w' = 0$ we can again show that the algorithm does not converge to a solution weight vector. In particular, the first search step gives the beam $\{D,B\}$ and since B is on the target path, we do not update the weights and generate a new beam $\{G,H\}$ by expanding the node D. At this point there are no target nodes in the beam and the weights are updated as follows

$$\begin{aligned} w' &= w' + f(B) - 0.5[f(G) + f(H)] \\ &= w' + [1,1] - [1,1] \\ &= w' \end{aligned}$$

showing that the algorithm has converged to $w' = 0$, which is not a solution to the problem.

Thus, we have shown that a positive search margin does not guarantee convergence for either LaSO-BR or LaSO-BST. This counter example also applies to the original LaSO algorithm, which is quite similar to LaSO-BST.

4.3 Convergence Under Stronger Notions of Margin

Given that linear separability, or equivalently a positive search margin, is not sufficient to guarantee convergence we consider a stronger notion of margin, the *level margin*, which measures by how much the target nodes are ranked above (or below) other non-target nodes at the same search level.

Definition 11 (Level Margin) *The level margin of a weight vector w for a training set is defined as $\gamma = min_{\{(v^*,v)\}}(w \cdot f(v^*) - w \cdot f(v))$, where the set $\{(v^*,v)\}$ contains any pair such that v^* is a target node at some depth j and v can be reached in j search steps from the initial search node—that is, v^* and v are at the same level.*

For breadth-first beam search, a positive level margin for w implies a positive search margin, but not necessarily vice versa, showing that level margin is a strictly stronger notion of separability. The following result shows that a positive level margin is sufficient to guarantee convergence of LaSO-BR. Throughout we will let R be a constant such that for all training instances, for all nodes v and v', $\|f(v) - f(v')\| \leq R$. The proof of this result follows similar lines as the Perceptron convergence proof for standard classification problems Rosenblatt (1962).

Theorem 12 *Given a dead-end free training set such that there exists a weight vector w with level margin $\gamma > 0$ and $\|w\| = 1$, LaSO-BR will converge with a consistent weight vector after making no more than $(R/\gamma)^2$ weight updates.*

Proof First note that the dead-end free property of the training data can be used to show that unless the current weight vector is a solution it will eventually trigger a "meaningful" weight update (one where the candidate set contains target nodes).

Let w^k be the weights before the k'th mistake is made. Then $w^1 = 0$. Suppose the k'th mistake is made for the training data $\langle S_i, P_i \rangle$, when $B \cap P_{i,j} = \emptyset$. Here, $P_{i,j}$ is the j'th element of P_i, B is the beam generated at depth j for S_i and C is the candidate set from which B is selected. Note that C is generated by expanding all nodes in the previous beam and at least one of them is in $P_{i,j-1}$. With the dead-end free property, we are guaranteed that $C' = P_{i,j} \cap C \neq \emptyset$. The occurrence of the mistake indicates that, $\forall v^* \in P_{i,j} \cap C, v \in B$, $w^k \cdot f(v^*) \leq w^k \cdot f(v)$, which lets us derive an upper bound for $\|w^{k+1}\|^2$.

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

where the first equality follows from the definition of the perceptron-update rule, the first inequality follows because $w^k \cdot (f(v^*) - f(v)) < 0$ for all $v^* \in C', v \in B$, and the second inequality follows from the definition of R. Using this upper-bound we get by induction that

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

Suppose there is a weight vector w such that $||w|| = 1$ and w has a positive level margin, then we can derive a lower bound for $w \cdot w^{k+1}$.

$$\begin{aligned} w \cdot w^{k+1} &= w \cdot w^k + w \cdot (\frac{\sum_{v^* \in C'} f(v^*)}{|C'|} - \frac{\sum_{v \in B} f(v)}{b}) \\ &= w \cdot w^k + \frac{\sum_{v^* \in C'} w \cdot f(v^*)}{|C'|} - \frac{\sum_{v \in B} w \cdot f(v)}{b} \\ &\geq w \cdot w^k + \gamma. \end{aligned}$$

This inequality follows from the definition of the level margin γ of the weight vector w.

By induction, we get that $w \cdot w^{k+1} \geq k\gamma$. Combining this result with the above upper bound on $||w^{k+1}||$ and the fact that $||w|| = 1$ we get that

$$1 \geq \frac{w \cdot w^{k+1}}{||w|| \, ||w^{k+1}||} \geq \sqrt{k}\frac{\gamma}{R} \Rightarrow k \leq \frac{R^2}{\gamma^2}.$$

Without the dead-end free property, LaSO-BR might generate a candidate set that contains no target nodes, which would allow for a mistake that does not result in a weight update. However, for a dead-end free training set, it is guaranteed that the weights will be updated if and only if a mistake is made. Thus, the mistake bound is equal to the bound on the weight updates. ■

Note that for the example search space in Figure 9 there is no weight vector with a positive level margin since the final layer contains target and non-target nodes with identical weight vectors. Thus, the non-convergence of LaSO-BR on that example is consistent with the above result. Unlike LaSO-BR, LaSO-BST and LaSO do not have such a guarantee since their beams can contain nodes from multiple levels. This is demonstrated by the following counter example.

Counter Example 2 *We give a training set for which the existence of a w with positive level margin does not guarantee convergence for LaSO-BST. Consider a single training example with the search space in Figure 10, single target path $P = (\{A\}, \{B\}, \{E\})$, and preference ordering $C < B < E < F < G < D$.*

Given the weight vector $w = [2\gamma, \gamma]$, the level margin of w is equal to γ. However, starting with $w' = 0$ and running LaSO-BST the first search step gives the beam $\{D, B\}$. Since B is on the target path, we get the new beam $\{G, F\}$ by expanding the node D. This beam does not contain a target node, which triggers the following weight update:

$$\begin{aligned} w' &= w' + f(B) - [f(F) + f(G)]/2 \\ &= w' + [1,0] - [1,0] \\ &= w'. \end{aligned}$$

Since w' does not change the algorithm has converged to $w' = 0$, which is not a solution to this problem. This shows that a positive level margin is not sufficient to guarantee the convergence of LaSO-BST. The same can be shown for the original LaSO.

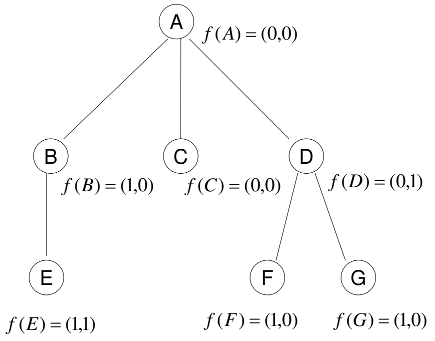

Figure 10: Counter example to convergence under positive level margin.

In order to guarantee convergence of LaSO-BST, we require an even stronger notion of margin, *global margin*, which measures the rank difference between any target node and any non-target node, regardless of search space level.

Definition 13 (Global Margin) *The global margin of a weight vector w for a training set is defined as $\gamma = min_{\{(v^*,v)\}}(w \cdot f(v^*) - w \cdot f(v))$, where the set $\{(v^*,v)\}$ contains any pair such that v^* is any target node and v is any non-target node in the search space.*

Note that if w has a positive global margin then it has a positive level margin. The converse is not necessarily true. The global margin is similar to the common definitions of margin used to characterize the convergence of linear perceptron classifiers (Novikoff, 1962).

To ensure convergence of LaSO-BST we also assume that the search spaces are all finite trees. This avoids the possibility of infinite best-first beam trajectories that never terminate at a goal node. Tree structures are quite common in practice and it is often easy to transform a finite search space into a tree. The structured classification experiments of Daumé III and Marcu (2005) and our own automated experiments both involve tree structured spaces.

Theorem 14 *Given a dead-end free training set of finite tree search spaces such that there exists a weight vector w with global margin $\gamma > 0$ and $\|w\| = 1$, LaSO-BST will converge with a consistent weight vector after making no more than $(R/\gamma)^2$ weight updates.*

The proof is similar to that of Theorem 12 except that the derivation of the lower bound makes use of the global margin and we must verify that the restriction to finite tree search spaces guarantees that each iteration of LaSO-BST will terminate with a goal node being reached. We were unable to show convergence for the original LaSO algorithm even under the assumptions of this theorem.

In summary, this section has introduced three different notions of margin: search margin, level margin, and global margin. Both algorithms converge for a positive global margin, which implies a positive search margin and a positive level margin. For LaSO-BR, but not LaSO-BST, convergence is guaranteed for a positive level margin, which implies a positive search margin. This shows that LaSO-BR converges under a strictly weaker notion of margin than LaSO-BST due to the fact that the ranking decisions of breadth-first search are restricted to nodes at the same level of the search space, as opposed to best-first search. This suggests that it may often be easier to define effective feature spaces for the breadth-first paradigm. Finally, a positive search margin corresponds exactly to linear separability, but is not enough to guarantee convergence for either algorithm. This is in contrast to results for linear classifier learning, where linear separability implies convergence of perceptron updates.

4.4 Convergence for Ambiguous Training Data

Here we study convergence for linearly inseparable training data. Inseparability is often the result of training-data ambiguity, in the sense that many "good" solution paths are not included as target paths. For example, this is common in automated planning where there can be many (nearly) optimal solutions, many of which are inherently identical (e.g., differing in the orderings of unrelated actions). It is usually impractical to include all solutions in the training data, which can make it infeasible to learn a ranking function that strictly prefers the target paths over the inherently identical paths not included as targets. In these situations, the above notions of margin will all be negative. Here we consider the notion of *beam margin* that allows for some amount of ambiguity, or inseparability.

For each instance $\langle S_i, P_i \rangle$, where $S_i = \langle I_i, s_i(\cdot), f(\cdot), <_i \rangle$ and $P_i = (P_{i,1}, P_{i,2}, \ldots, P_{i,d_i})$, let D_{ij} be the set of nodes that can be reached in j search steps from I_i. That is, D_{ij} is the set of all possible non-target nodes that could be in beam $B_{i,j}$. 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 15 (Beam Margin) *A weight vector w has beam margin (b', δ_1, δ_2) on a training set $\{\langle S_i, P_i \rangle\}$, if for each i, j there is a set $D'_{ij} \subseteq D_{ij}$ such that $|D'_{ij}| \leq b'$ and*

$$\forall v^* \in P_{i,j}, v \in D_{ij} - D'_{ij}, \;\; w \cdot f(v^*) - w \cdot f(v) \geq \delta_1 \textit{ and,}$$
$$\forall v^* \in P_{i,j}, v \in D'_{ij}, \;\; \delta_1 > w \cdot f(v^*) - w \cdot f(v) \geq -\delta_2.$$

A weight vector w has beam margin (b', δ_1, δ_2) if at each search depth it ranks the target nodes better than most other non-target nodes (those in $D_{ij} - D'_{ij}$) by a margin of at least δ_1, and ranks at most b' non-target nodes (those in D'_{ij}) better than the target nodes by a margin no greater than δ_2. Whenever this condition is satisfied we are guaranteed that a beam search of width $b > b'$ guided by w will solve all of the training problems. The case where $b' = 0$ corresponds to the level margin, where the data is separable. By allowing $b' > 0$ we can consider cases where there is no "dominating" weight vector that ranks all targets better than all non-targets at the same level. The following result shows that for a large enough beam width, which is dependent on the beam margin, LaSO-BR will converge to a consistent solution.

Theorem 16 *Given a dead-end free training set, if there exists a weight vector w with beam margin (b', δ_1, δ_2) and $\|w\| = 1$, then for any beam width $b > (1 + \delta_2/\delta_1) b' = b^*$, LaSO-BR will converge with a consistent weight vector after making no more than $(R/\delta_1)^2 \left(1 - b^* b^{-1}\right)^{-2}$ weight updates.*

Proof Let w^k be the weights before the k'th mistake is made, so that $w^1 = 0$. Suppose that the k'th mistake is made when $B \cap P_{i,j} = \emptyset$ where B is the beam generated at depth j for the ith training instance. We can derive the upper bound of $\|w^{k+1}\|^2 \leq kR^2$ as in the proof of Theorem 12.

Next we derive a lower bound on $w \cdot w^{k+1}$. Denote by $B' \subseteq B$ the set of nodes in the beam such that $\delta_1 > w \cdot (f(v^*) - f(v)) \geq -\delta_2$ and let $C' = P_{i,j} \cap C$. By the definition of beam margin, we have $|B'| < b'$.

$$
\begin{aligned}
w \cdot w^{k+1} &= w \cdot w^k + w \cdot \Big(\frac{\sum_{v*\in C'} f(v^*)}{|C'|} - \frac{\sum_{v\in B} f(v)}{b}\Big) \\
&= w \cdot w^k + w \cdot \sum_{v\in B-B'} \frac{\frac{\sum_{v*\in C'} f(v^*)}{|C'|} - f(v)}{b} \\
&\qquad + w \cdot \sum_{v\in B'} \frac{\frac{\sum_{v*\in C'} f(v^*)}{|C'|} - f(v)}{b} \\
&\geq w \cdot w^k + \frac{(b-b')\delta_1}{b} - \frac{b'\delta_2}{b}.
\end{aligned}
$$

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 $1 \geq \frac{w \cdot w^{k+1}}{\|w\|\|w^{k+1}\|} \geq \sqrt{k}\frac{(b-b')\delta_1 - b'\delta_2}{bR}$. The mistake bound follows by noting that $b > b^*$ and algebra.

Similar to Theorem 12, the dead-end free property of the training set guarantees that the mistake bound is equal to the bound on the weight updates. ■

Note that when there is a positive level margin (i.e., $b' = 0$), the mistake bound here reduces to $(R/\delta_1)^2$, which does not depend on the beam width and matches the result for separable data. This is also the behavior when $b >> b^*$.

An interesting aspect of this result is that the mistake bound depends on the beam width. Rather, all of our previous convergence results were independent of the beam width and held even for beam width $b = 1$. Thus, those previous results did not provide any formalization of the intuition that the learning problem will often become easier as the beam width increases, or equivalently as the amount of search increases. Indeed, in the extreme case of exhaustive search, no learning is needed at all, whereas for $b = 1$ the ranking function has little room for error.

To get a sense for the dependence on the beam width consider two extreme cases. As noted above, for very large beam widths such that $b >> b^*$, the bound becomes $(R/\delta_1)^2$. On the other extreme, if we assume $\delta_1 = \delta_2$ and we use the smallest possible beam width allowed by the theorem $b = 2b' + 1$, then the bound becomes $((2b'+1)R/\delta_1)^2$, which is a factor of $(2b'+1)^2$ larger than when $b >> b'$. This shows that as we increase b (i.e., the amount of search), the mistake bound decreases, suggesting that learning becomes easier, agreeing with intuition.

It is also possible to define an analog to the beam margin for best first beam search. However, in order to guarantee convergence, the conditions on ambiguity would be relative to the global state space, rather than local to each level of the search space.

5. Application to Automated Planning

In this section, we present an empirical evaluation of beam-search learning in the context of automated planning. We first give related background, followed by the technical details regarding our application to automated planning. Then, we present the experimental results.

5.1 Background

Here we give background related to automated planning, the problem of learning to plan, and prior related work in the area of learning to plan.

5.1.1 AUTOMATED PLANNING

Planning is a subfield of artificial intelligence that studies algorithms for selecting sequences of actions in order to achieve goals. In this work, we consider planning domains and planning problems described using the STRIPS fragment of the planning domain description language (PDDL) (McDermott, 1998), which we now outline.

A planning domain $\mathcal{D}$ defines a set of possible actions $\mathcal{A}$ and a set of world states $\mathcal{W}$ 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 $\mathrm{Pre}(a)$, 3) two sets of state facts $\mathrm{Add}(a)$ and $\mathrm{Del}(a)$ representing the add and delete effects respectively. An action a is applicable to a world state ω iff $\mathrm{Pre}(a) \subseteq \omega$, and the application of an (applicable) action a to ω results in the new state $\omega' = (\omega \setminus \mathrm{Del}(a)) \cup \mathrm{Add}(a)$. That is, the application of an action adds the facts in the add list to the state and deletes facts in the delete list.

Given a planning domain, a planning problem is a tuple (ω, A, g), where $A \subseteq \mathcal{A}$ is a set of actions, $\omega \in \mathcal{W}$ 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 $\langle a_1, \ldots, a_l \rangle$, where the sequential application of the sequence starting in state ω leads to a goal state ω' where $g \subseteq \omega'$. 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.

Figure 1 shows an example of the search space corresponding to a problem from the Blocksworld planning domain. Here, the initial state is described by the facts

$$\begin{aligned}\omega_0 = \{&clear(A), clear(B), clear(C), clear(D), ontable(A),\\ &ontable(B), ontable(C), ontable(D), armempty\}.\end{aligned}$$

An example action from the domain is $pickup(A)$ with the following definition:

$$\begin{aligned}Pre(pickup(A)) &= \{clear(A), ontable(A), armempty\}\\ Add(pickup(A)) &= \{holding(A)\}\\ Del(pickup(A)) &= \{clear(A), ontable(A), armempty\}.\end{aligned}$$

Note that the precondition of this action is satisfied in ω_0 and hence can be applied from ω_0, which would result in the new state

$$\omega_1 = \{holding(A), clear(B), clear(C), clear(D), ontable(B), ontable(C), ontable(D)\}.$$

If the goal of the planning problem is $g = \{on(C,D), on(B,A), clear(C), clear(B)\}$, then one solution for the problem, as shown in Figure 1, is the action sequence $\langle pickup(B), stack(B,A)$, $pickup(C)$, $stack(C,D)\rangle$.

There has been much recent progress in automated planning. One of the most successful approaches, and the one most relevant to this paper, is to solve planning problems using forward state-space search guided by powerful domain-independent planning heuristics. A number of recent state-of-the-art planners have followed this approach including HSP (Bonet and Geffner, 1999), FF (Hoffmann and Nebel, 2001), and AltAlt (Nguyen et al., 2002) to name just a few.

5.1.2 LEARNING TO PLAN

It is common for planning systems to be asked to solve many problems from a particular domain. For example, the bi-annual international planning competition is organized around a number of planning domains and includes many problems of varying difficulty from each domain. Given that problems from the same domain share significant structure, it is natural to attempt to learn from past experience in a domain in order to solve future problems from the same domain more efficiently. However, most state-of-the-art planning systems do not have any such learning capability and rather solve each problem from the domain as if it were the first problem ever encountered by the planner. The goal of our work is to develop the capability for a planner to learn domain-specific knowledge in order to improve performance in a target domain of interest.

More specifically, 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, or ranking functions, 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.

5.1.3 PRIOR WORK

There has been a long history of work on learning-to-plan, originating at least back to the original STRIPS planner (Fikes et al., 1972), which learned triangle tables or macros that could later be exploited by the planner. For a collection and survey of work on learning in AI planning see Minton (1993) and Zimmerman and Kambhampati (2003).

A number of learning-to-plan systems have been based on the explanation-based learning (EBL) paradigm, for example, Minton et al. (1989) among many others. EBL is a deductive learning approach, in the sense that the learned knowledge is provably correct. Despite the relatively large effort invested in EBL research, the best approaches typically did not consistently lead to significant gains, and even hurt performance in many cases. A primary way that EBL can hurt performance is by learning too many, overly specific control rules, which results in the planner spending too much time simply evaluating the rules at the cost of reducing the number of search nodes considered. This problem is commonly referred to as the EBL utility problem (Minton, 1988).

Partly in response to the difficulties associated with EBL-based approaches, there have been a number of systems based on inductive learning, sometimes combined with EBL. The inductive approach involves applying statistical learning mechanisms in order to find common patterns that can distinguish between good and bad search decisions. Unlike EBL, the learned control knowledge typical does not have guarantees of correctness, however, the knowledge is typically more general and hence more effective in practice. Some representative examples of such systems include learning for partial-order planning (Estlin and Mooney, 1996), learning for planning as satisfiability (Huang et al., 2000), and learning for the Prodigy means-ends framework (Aler et al., 2002). While these systems typically showed better scalability than their EBL counterparts, the evaluations were typically conducted on only a small number of planning domains and/or small number of test problems. There is no empirical evidence that such systems are robust enough to compete against state-of-the-art non-learning planners across a wide range of domains.

More recently there have been several learning-to-plan systems based on the idea of learning reactive policies for planning domains (Khardon, 1999; Martin and Geffner, 2000; Yoon et al., 2002). These approaches use statistical learning techniques to learn policies, or functions, that map any state-goal pair from a given domain to an appropriate action. Given a good reactive policy for a domain, problems can be solved quickly, without search, by iterative application of the policy. Despite its simplicity, this approach has demonstrated considerable success. However, these approaches have still not demonstrated the robustness necessary to outperform state-of-the-art non-learning planners across a wide range of domains.

More closely related is work by La Rosa et al. (2007), which uses a case-based reasoning approach to obtained an improved heuristic for forward state-space search. It is likely that our weight learning approach could be combined with their system to harness the benefits of both approaches. The most closely related approach to our work is based on extending forward state-space search planners by learning improved heuristics (Yoon et al., 2006), an approach which is among the state-of-the-art among learning-based planners. That work focused on improving the relaxed plan length heuristic used by the state-of-the-art planner FF (Hoffmann and Nebel, 2001). Note that FF consists of two stages: an incomplete local search and a complete best first search. In particular, Yoon et al. (2006) applied 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 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 on the training data when used for search, the learner makes no attempt to correct the heuristic in response.

A primary motivation for this paper is to develop a heuristic learning mechanism that is more tightly integrated with the search process. Our LaSO-style algorithms for learning beam-search ranking functions do exactly that. Our learning approach can be viewed as error-driven in the sense that it directly attempts to correct errors as they arise in the search process, rather than attempting to precisely model the distance-to-goal. In many areas of machine learning, such error-driven methods have been observed to outperform their traditional passive counterparts. The experimental results presented here agree with that observation in a number of planning domains.

5.2 Experimental Setup

We present experiments in eight STRIPS domains: Blocksworld, Pipesworld, Pipesworld-with-tankage, PSR, Philosopher, DriverLog, Depots and FreeCell. All of these domains with the exception of Blocksworld were taken from the 3rd and 4th international planning competitions (IPC3 and IPC4). With only two exceptions, this is the same set of domains used to evaluate the approach of

Yoon et al. (2006), which is the only prior work that we are aware of for learning heuristics to improve forward state-space search in automated planning. The difference between our set of domains and theirs is that we include Blocksworld, while they did not, and we do not include the Optical Telegraph domain, while they did. Our reason for not showing results for Optical Telegraph is that none of the systems we evaluated were able to solve any of the problems.[1]

5.2.1 DOMAIN PROBLEM SETS

For each domain we needed to create a set of training problems and testing problems on which the learned heuristics would be trained and evaluated. In Blocksworld, all problems were generated using the BWSTATES generator (Slaney and Thiébaux, 2001), which produces random Blocksworld problems. Thirty problems with 10 or 20 blocks were used as training data, and 30 problems with 20, 30, or 40 blocks were used for testing. For DriverLog, Depots and FreeCell, the first 20 problems are taken from IPC3 and we generated 30 more problems of varying difficulty to arrive at a set of 50 problems, roughly ordered by difficulty. For each domain, we used the first 15 problems for training and the remaining 35 for testing. The other four domains are all taken from IPC4. Each domain includes 50 or 48 problems, roughly ordered by difficulty. In each case, we used the first 15 problems for training and the remaining problems for testing.

5.2.2 SEARCH SPACE DEFINITION

We now describe the mapping between the planning problem described in Section 5.1.1 and the general search spaces described in Section 2, which were the basis for describing our algorithms. Recall that a general search space is a tuple $\langle I, s(\cdot), f(\cdot), < \rangle$ giving the initial state, successor function, feature function, and preference ordering respectively. In the context of planning each search node is a state-goal pair (ω, g), where ω can be thought of as the current world state, g is the current goal, and both are represented as sets of facts. Note that it is important that nodes contain both state and goal information, rather than just state information, since the evaluation/ranking of a search node depends on how good ω is with respect to the particular goal g. The initial search node I is equal to (ω_0, g), where ω_0 is the initial state of the planning problem and g is the problem's goal. The successor function s maps a search node (ω, g) to the set of all nodes of the form (ω', g) where ω' is a state that can be reached from ω via the application of some action whose preconditions are satisfied in ω. Note that according to this definition all nodes in a search space contain the same goal component. The feature function $f((\omega,g)) = (f_1((\omega,g)), \ldots, f_m((\omega,g)))$ can be any function over world states and goals. The particular functions we use in this work are describe later in this section. Finally, the preference ordering $<$ is simply the default ordering used by the planner FF, which is the planner our implementation is based on.

5.2.3 TRAINING DATA GENERATION

The LaSO-style algorithms learn from target solution paths, which requires that we generate solution plans for all of the training problems. To do this, for each training problem, we selected the shortest plan out of those found by running the planner FF and beam search with various large beam

[1] The results in Yoon et al. (2006) indicated that their linear regression learning method was effective in Optical Telescope. Our implementation of linear regression, however, was unable to solve any of the problems. After investigating this difference, we found that it is due to a subtle difference in the way that ties are broken during forward state-space search, indicating that the linear regression method was not particularly robust in this domain.

widths guided by FF's relaxed-plan heuristic. The resulting plans are totally ordered sequences of actions and one could simply label each training problem by its corresponding sequence of actions. However, in many cases, it is possible to produce equivalent plans by permuting the order of certain actions in the totally ordered plans. That is, there are usually many other equivalent totally ordered plans. Thus, including only the single plan found via the above approach in the training data results in significant ambiguity in the sense described in Section 4.4.

In order to help reduce the ambiguity it is desirable to try to include as many equivalent plans as possible as part of the target plan set for a particular problem. To do this, instead of using just a single totally ordered solution plan in the training data for each problem, we transform each such totally ordered plan into a partial-order plan, which contains the same set of actions but only includes action-ordering constraints that appear to be necessary. Finding minimal partial-order plans from total-order plans is an NP-hard problem and hence we use the heuristic algorithm described in Veloso et al. (1991). For each training problem, the resulting partial-order plan provides an implicit representation for a potentially exponentially large set of solution trajectories. By using these partial-order plans as the labels for our training problems we can significantly reduce the ambiguity in the training data. In preliminary experiments, the performance of our learning algorithms improved in a number of domains when using training data that included the partial-order plans rather than the original total-order plans.

5.2.4 HEURISTIC REPRESENTATION AND DOMAIN FEATURES

We consider learning heuristic functions that are represented as weighted linear combinations of features, that is, $H(v) = \Sigma_i w_i \cdot f_i(v)$ where v is a search node, f_i is a feature of search nodes, and w_i is the weight of feature f_i. One of the challenges with this representation is to define a generic feature space from which features can be selected for each domain. 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 et al. (2008) that defined such a feature space using a first-order language.

Each feature in the above space is defined by a taxonomic class expression, which represents a set of constants/objects in the planning domain. For example, a simple taxonomic class expression for the Blocksworld planning domain is *clear*, which represents the set of blocks that are currently clear, that is, the set of blocks x such that $clear(x) \in \omega$ where the current search node is $v = (\omega, g)$. The respective feature value represented by a class expression is equals to the cardinality of the class expression when evaluated at a search node. For example, if we let f_1 be the feature represented by the class expression *clear* then $f_1((\omega, g))$ is simply the number of clear blocks in ω. So in the example states from Section 5.1.1, $f_1(v_0) = f_1((\omega_0, g)) = 4$ and $f_1(v_1) = f_1((\omega_1, g)) = 3$. A more complex example for this problem is $clear \cap gclear$, which represents the set of blocks that are clear in both the current state and the goal, that is, the set containing any block x such that $clear(x) \in \omega$ and $clear(x) \in g$. If f_2 represents the feature corresponding to this expression then in the example states from 5.1.1 we get that $f_2(v_0) = 2$ and $f_2(v_1) = 2$.

Since our work in this paper is focused on weight learning, we refer to Yoon et al. (2008) for the full definition of the taxonomic feature language. Here we simply use a set of taxonomic features that have been automatically learned in prior work (Yoon et al., 2008) and tune their weights. In our experiments, this prior work gave us 15 features in Blocksworld, 35 features in Pipesworld, 11 features in Pipesworld-with-tankage, 54 features in PSR, 19 features in Philosopher, 22 features in

DriverLog, 3 features in Depot and 3 features in FreeCell. In all cases, we include FF's relaxed-plan-length heuristic as an additional feature.

5.3 Experimental Results

For each domain, we use LaSO-BR to learn weights with a learning rate of 0.01 for beam widths 1, 10, 50, and 100 and we will denote LaSO-BR run with beam width b by LaSO-BR$_b$. The maximum number of LaSO-BR iterations was set to 5000. In the evaluation procedure, we set a time cut-off of 30 CPU minutes per problem and considered a problem to be unsolved if a solution was not found within the cut-off.

In preliminary work, we also tried to apply LaSO-BST to our problems. However, this turned out to be an impractical approach due to the large potential search depths of these problems. In particular, we found that in many cases LaSO-BST would become stuck processing training examples, in the sense that it would neither update the weights nor make progress with respect to following the target trajectories. This typically occurred because LaSO-BST would maintain an early target node in the beam and thus not trigger a weight update, but at the same time would not progress to include deeper nodes on the target trajectories and instead explore paths off the target trajectories. To help remedy this behavior, we experimented with a variant of LaSO-BST that forces progress along the target trajectories after a specified number of search steps. For the Blocksworld planning domain and preliminary experiments in the other domains, we found that the results tended to improve compared to the original LaSO-BST, but still were not competitive with LaSO-BR. Thus for the experiments reported below we focus on LaSO-BR.

Note that the experiments in Daumé III and Marcu (2005) for structured classification produced good results using an algorithm very similar to LaSO-BST. There, however, the search spaces have small maximum depths (e.g., the length of a sentence), which apparently helped to avoid the problem we experienced here.

5.3.1 TRAINING TIME

Figure 11 gives the average training time required by LaSO-BR per iteration in each of our domains for four different beam widths. Note that Pipesworld was the only domain for which LaSO-BR converged to a consistent weight vector using a learning beam width 100. For all other training sets LaSO-BR never converged and thus terminated after 5000 iterations. The training time varies widely across the domains and depends on various factors including: number of features, number of actions, number of state predicates, and the number and length of target trajectories per training example. As expected the training times increase with the training beam width across the domains. It is difficult, however, to predict the relative times between different domains due to the complicated interactions among the above factors. Note that while these training times can be significant in many domains, the cost of training needs to only be paid once and then it is amortorized over all future problems. Furthermore, as we can observe later in the experimental results, a small beam width of 10 typically performs as well as larger widths.

5.3.2 DESCRIPTION OF TABLES

Before presenting our results we will first provide an overview of the information contained in our results tables. Figure 12 compares the performance of LaSO-BR$_{10}$ to three other algorithms,

Domain	$b = 1$	$b = 10$	$b = 50$	$b = 100$
Blocksworld	3	15	66	128
Pipesworld	1	4	13	24
Pipesworld-with-tankage	3	17	76	149
PSR	53	127	403	690
Philosopher	3	24	121	260
DriverLog	1	5	22	44
Depots	5	32	160	320
FreeCell	10	68	315	654

Figure 11: The average training time required by LaSO-BR per iteration for all training instances (seconds).

- LEN : beam search using FF's relaxed plan length heuristic
- U : beam search using a heuristic with uniform weights for all features
- LR : beam search using the heuristic learned from linear regression following the approach in Yoon et al. (2006).

We selected LaSO-BR$_{10}$ here because its performance is on par or better than other training beam widths. Note that in practice one could select the best beam width to use via cross-validation with a validation set of problems.

There is one table for each of our domains and each column in the tables is labeled by the algorithm used to generate the results. The rows correspond to the beam width used to generate the results on the testing problems, with the last row corresponding to using full best-first search (BFS) with an infinite beam width, which is the native search procedure used by FF. The columns are divided into three sets. The first four data columns labeled "Problems solved" give the number of problems solved using the testing beam width corresponding to the row, where a problem is considered solved if a solution is found within 30 minutes. The second set of columns labeled "Median plan length" gives the median length of solutions to the planning problems that were solved. The last 4 columns labeled "Median runtime " give the median runtime of each solver on the problems it solved. So, for example, the table shows that the heuristic learned via LaSO-BR$_{10}$ solves 26 Blocksworld test problems with a median solution length of 139 and a median runtime of 58.8 seconds using a testing beam width of 50, and solved 19 problems with a median solution length of 142 and a median runtime of 20.9 seconds using BFS.

Figure 13 is similar in structure to Figure 12 but compares the performance of heuristics learned using LaSO-BR with a variety of training beam widths and evaluated using a variety of testing beam widths. Only the number of problems solved and the median length of solutions that are found are considered here. For example, the upper left-most data point gives the number of problems solved using a learning beam width of 1 and a testing beam width of 1, while the first entry in the last column gives the median plan length of solved problems when learning with beam width 100 and testing with beam width 1.

Blocksworld												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	13	0	11	**24**	3318	-	938	499	12.3	-	3.4	4.5
10	22	0	19	**24**	449	-	120	293	15.9	-	9.9	25.1
50	20	0	19	**26**	228	-	64	139	37.5	-	10.4	58.8
100	19	0	20	**24**	110	-	67	144	52.0	-	42.8	110.3
500	17	0	**23**	17	80	-	74	96	74.2	-	379.1	133.2
BFS	5	0	13	**19**	80	-	76	142	3.7	-	18.0	20.9

Pipesworld												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	11	13	8	**16**	114	651	2476	2853	0.6	3.2	21.7	17.8
10	17	17	21	**23**	112	360	194	222	15.6	13.2	10.2	15.8
50	18	19	21	**26**	34	167	89	80	9.4	42.8	25.5	27.8
100	18	16	21	**24**	32	39	60	62	19.7	12.0	23.3	39.3
500	21	18	21	**25**	30	33	31	53	62.9	58.3	101.8	95.1
BFS	**15**	7	7	**15**	44	54	42	54	35.5	1.1	3.1	1.3

Pipesworld-with-tankage												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	6	4	2	**7**	119	416	1678	291	8.0	18.2	92.1	8.4
10	6	8	**9**	8	68	603	399	117	70.2	256.5	125.6	33.4
50	6	5	6	**11**	61	111	94	122	358.4	281.4	186.1	116.3
100	5	4	5	**8**	54	105	43	55	482.4	279.4	255.5	190.6
500	5	6	4	**10**	42	97	41	76	938.5	586.1	210.7	492.0
BFS	**5**	3	2	3	59	60	126	100	431.2	17.1	935.7	22.0

PSR												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	0	0	0	0	-	-	-	-	-	-	-	-
10	1	**20**	13	13	516	157	151	193	840.1	367.9	186.6	492.4
50	13	**17**	16	10	99	109	99	97	685.3	658.2	890.4	802.4
100	13	**15**	13	6	103	89	89	85	999.4	1121.9	1215.0	643.1
500	**4**	**4**	2	1	55	59	48	39	1035.6	1157.6	689.1	423.9
BFS	13	0	**21**	**21**	89	-	131	141	686.7	-	290.8	526.0

Philosopher												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	0	**33**	**33**	**33**	-	363	363	363	-	12.5	18.1	13.3
10	0	**33**	**33**	11	-	363	363	1154	-	121.3	171.0	101.3
50	0	6	**23**	13	-	215	308	1579	-	77.6	387.4	825.1
100	0	16	**18**	6	-	292	281	1076	-	489.0	507.6	911.1
500	0	**7**	**7**	2	-	220	220	745	-	792.3	844.6	1280.7
BFS	0	**33**	**33**	0	-	363	363	-	-	9.5	329.8	-

DriverLog												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	0	0	0	**8**	-	-	-	6801	-	-	-	364.2
10	3	0	0	**12**	789	-	-	1439	967.8	-	-	781.3
50	4	8	0	**12**	108	177	-	541	1199.3	457.6	-	998.5
100	1	**11**	0	**11**	98	147	-	275	1398.9	737.9	-	1131.6
500	0	**3**	0	1	-	86	-	94	-	1780.2	-	1237.1
BFS	**6**	2	0	1	162	181	-	138	1249.7	555.5	-	125.4

Depots												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	1	1	2	**3**	462	790	411	790	3.9	6.5	3.8	6.7
10	4	1	4	**6**	195	28	981	3295	38.7	2.4	93.1	594.8
50	3	4	5	**6**	25	511	51	467	22.4	912.8	17.3	156.0
100	4	**7**	3	**7**	232	157	26	207	554.9	669.4	45.6	189.9
500	5	4	6	**11**	38	62	39	53	274.2	351.2	422.7	477.8
BFS	2	2	**3**	2	46	48	33	48	292.4	809.3	14.2	386.8

FreeCell												
	Problems solved				Median plan length				Median runtime (seconds)			
b	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$	LEN	U	LR	LaSO-BR$_{10}$
1	5	7	4	**9**	96	120	146	123	12.2	21.5	13.8	14.0
10	20	**22**	19	21	82	117	243	89	99.7	165.2	305.2	91.9
50	23	**24**	12	19	65	73	102	66	456.2	503.4	619.0	367.9
100	20	18	7	**21**	65	63	70	65	723.5	720.9	673.4	796.1
500	3	3	2	**4**	53	55	59	55	1400.0	1418.8	1518.5	1431.8
BFS	**23**	20	12	20	78	87	111	97	102.1	77.9	238.4	92.3

Figure 12: Experimental results for different planners. For each domain, we show the number of solved problems, the median plan length and median runtime of the solved problems. A dash in the table indicates that the planner was unable to solve any of the problems.

5.3.3 PERFORMANCE ACROSS TESTING BEAM WIDTHS

From Figure 12, in general, for all algorithms (learning and non-learning) we see that as the testing beam width begins to increase the number of solved problems and runtime increase and solution lengths improve. However, at some point as the beam width continues to increase the number of solved problems typically decreases. This behavior is typical for beam search, since as the testing beam width 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 as the beam width becomes large.

The median runtime typically increases as the test beam width increases, because more search nodes need to be evaluated. However, it is not always the case. The number of search nodes that are going to be evaluated also depends on the plan length. For example, while using LEN in the Depots planning domain, the median runtime of beam width 50 is smaller than that of beam width 10, because the median plan length improves from 195 to 25. Also note that it is not necessarily true that the plan lengths are strictly non-increasing with testing beam width. With large testing 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 testing beam widths.

5.3.4 LaSO-BR$_{10}$ VERSUS NO LEARNING

From Figure 12, we see that compared to LEN, the heuristic learned by LaSO-BR$_{10}$ tends to significantly improve the performance of beam search, especially for small beam widths. For example, in Blocksworld with beam width 1, LaSO-BR$_{10}$ solves almost twice as many problems as LEN. The median plan length has also been reduced significantly for beam width 1. As the beam width increases the gap between LaSO-BR$_{10}$ and LEN decreases but LaSO-BR$_{10}$ still solves more problems with comparable solution quality. In Pipesworld, LaSO-BR$_{10}$ has the best performance gap with beam width 50, solving 8 more problems than LEN. As the beam width increases, again the performance gap decreases, but LaSO-BR$_{10}$ consistently solves more problems than LEN. In this domain, the median plan lengths of LEN tend to be better, though a direct comparison of these lengths is not exactly fair since LaSO-BR$_{10}$ solves more problems, which are often the harder problems that result in longer plans. The trends with respect to number of solved problems are similar in other domains, with the exception of PSR and FreeCell. In PSR, LEN solves slightly more problems than LaSO-BR$_{10}$ at large beam widths. In FreeCell, LaSO-BR$_{10}$ is better than LEN for most case except for beam width 50.

These results show that LaSO-BR$_{10}$ is able to learn heuristics that significantly improve on the state-of-the-art heuristic LEN when using beam search. In general, the best performance was achieved for small beam widths close to those used for training, which is beneficial in terms of time and memory efficiency. Note that in practice one could use a validation set of problems in order to select the best combination of training beam width and testing beam width for a given domain. This is particularly natural in our current setting where our goal is to perform well relative to problems drawn from a given problem generator, in which case we can easily draw both training and evaluation problem sets.

5.3.5 COMPARING LaSO-BR$_{10}$ WITH LINEAR REGRESSION

To compare with prior passive heuristic learning work we learned weights using linear regression following the approach of Yoon et al. (2006). To our knowledge this is the only previous system that addresses the heuristic learning problem in the context of forward state-space search in automated planning. In these experiments we used the linear regression tool available under Weka. The results for the resulting learned linear-regression heuristics are shown in the columns labeled LR in Figure 12.

For Blocksworld, LR solves fewer problems than LaSO-BR$_{10}$ with beam widths smaller than 500 but solves more problems than LaSO-BR$_{10}$ with beam width 500. The median plan length tends to favor LR except for the smallest beam width $b = 1$. For Pipesworld, DriverLog and Depots, LaSO-BR$_{10}$ always solves more problems than LR, with plan length again favoring LR to varying degrees. In Pipesworld-with-tankage, LaSO-BR$_{10}$ is better than LR for most case except for beam width 10, solving one less problem. In PSR and Philosopher, LR outperforms LaSO-BR$_{10}$ but LaSO-BR$_{10}$ achieves a comparable performance with small beam widths. In FreeCell, LaSO-BR$_{10}$ always solves more problems than LR with improved plan length.

These results indicate that error-driven learning can significantly improve over prior passive learning (here regression) in a number of domains. Indeed, there appears to be utility in integrating the learning process directly in the search procedure. However, the results also indicate that in some cases our current error-driven training method can fail to converge to a good solution in cases where regression happens to work well.

5.3.6 EFFECTS OF LEARNING BEAM WIDTH

Figure 13 compares the performance of LaSO-BR with different learning beam widths. For most domains, the performance doesn't change much as the learning beam width changes. Even with learning beam width 1, LaSO-BR can often achieve performance on par with larger learning beam widths. For example, in Blocksworld, LaSO-BR$_1$ results in the best performance at most testing beam widths except for beam width 500. For the other domains, LaSO-BR$_{10}$ typically is close to the performance of the best learning beam width. In a number of cases we see that LaSO-BR$_{10}$ performs significantly better than LaSO-BR$_{100}$, which suggests that learning with smaller beam widths can have some practical advantages. One reason for this might be due to the additional ambiguity in the weight updates when using larger beam widths. In particular, the weight update equations involve averages of all target and non-target nodes in the beams. The effect of this averaging is to effectively mix the feature vectors of large numbers of search nodes together. In many cases there will be a wide variety of non-target nodes in the beam, and this mixing can increase the difficulty of uncovering key patterns, which we conjecture might increase the requirements on training iterations and examples. In cases where the features are rich enough to support successful beam search with small width, it is then likely that learning with smaller widths will be better given a fixed number of iterations and examples. Note that the feature space we have used in this work has been previously demonstrated (Fern et al., 2006) to be particularly well suited to Blocksworld, which is perhaps one reason that $b = 1$ performed so well in that domain.

Finally note that contrary to what we originally expected it is not typically the case that the best performance for a particular testing beam width is achieved when learning with that same beam width. Rather the relationship between learning and testing beam widths is quite variable. Note that for most domains LaSO-BR never converged to a consistent weight vector in our experiments, indicating that either the features were not powerful enough for consistency or the learning beam widths and/or number of iterations needed to be increased. In such cases, there is no clear technical reason to expect the best testing beam width to match the learning beam width. Thus, in general, we suggest the use of validation sets to select the best pair of learning and testing beam widths for a particular domain. Note that the lack of relationship between learning and test beam width is in contrast to that observed in Daumé III and Marcu (2005) for structured classification, where there appeared to be a small advantage to training and testing using the same width.

5.3.7 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 et al. (2006) and is the native search strategy used by FF.[2] These results are shown in the bottom row of each table in Figure 12 and 13.

In Blocksworld, Pipesworld, PSR, LaSO-BR$_{10}$ was as good or better than the other three algorithms. Especially in Blocksworld, LaSO-BR$_{10}$ solves 19 problems while LEN only solves 5 problems. In Philosopher, neither LEN nor LaSO-BR$_{10}$ solves any problem. LEN is the best in Pipesworld-with-tankage, DriverLog and FreeCell, and LR works best in Depots. But for Pipesworld-with-tankage, Depots and FreeCell, the performance of LaSO-BR$_{10}$ is very close to the best planner.

These results indicate that the advantage of error-driven learning over regression is not just restricted to beam search, but appears to extend to other search approaches. That is, by learning in the context of beam search it is possible to extract problem solving information that is useful in other contexts.

5.3.8 PLAN LENGTH

LaSO-BR can significantly improve success rate at small beam widths, which is one of our main goals. However, the plan lengths at small widths are quite suboptimal, which is typical of beam search. Ideally we would like to obtain these success rates without paying a price in plan length. We are currently investigating ways to improve LaSO-BR in this direction. However, we note that typically one of the primary difficulties of automated planning is to simply find a path to the goal. After finding such a path, if it is significantly sub-optimal, incomplete plan analysis or plan rewriting rules can be used to significantly prune the plan, for example, see Ambite et al. (2000). Thus, despite the long plan lengths, the improved success rate of LaSO-BR at small beam widths could provide a good starting point for a fast plan length optimization.

6. Summary and Future Work

This paper presented a detailed study of the problem of learning ranking functions for beam search with an application to automated planning. On the theoretical side we first studied the computational complexity of this learning problem, highlighting the main dimensions of complexity by identifying core tractable and intractable subclasses. Next, we studied the convergence of recent online learning algorithms for this problem. The results clarified convergence issues, correcting and extending

[2] FF actually uses two search strategies. In the first state it uses an incomplete strategy called enforced hill climbing. If that initial search does not find a solution then a best-first search is conducted.

Blocksworld								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	**27**	24	18	13	840	499	92	314
10	**27**	24	20	19	206	293	96	150
50	**27**	26	23	24	180	139	72	82
100	**25**	24	23	23	236	144	72	86
500	23	17	19	**24**	122	96	62	77
BFS	**21**	19	18	17	116	142	73	124

Pipesworld								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	16	16	**21**	15	1803	2853	1403	6958
10	**25**	23	23	21	227	222	179	270
50	25	**26**	25	22	74	80	119	75
100	**27**	24	23	22	146	62	104	47
500	23	**25**	20	21	60	53	61	37
BFS	14	**15**	13	8	59	54	103	42

Pipesworld-with-tankage								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	5	**7**	2	**7**	55	291	197	300
10	8	8	8	**10**	103	117	68	77
50	9	**11**	8	9	48	122	37	42
100	8	8	**10**	**10**	53	55	122	55
500	9	**10**	5	**10**	30	76	39	96
BFS	**6**	3	4	**6**	48	100	70	63

PSR								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	0	0	0	0	-	-	-	-
10	12	13	3	**14**	182	193	550	205
50	6	10	16	**17**	75	97	126	129
100	3	6	10	**13**	82	85	113	86
500	2	1	**4**	**4**	61	39	58	64
BFS	19	21	3	**25**	164	141	170	142

Philosopher								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	6	**33**	**33**	0	589	363	363	-
10	**19**	11	1	1	319	1154	451	1618
50	**13**	**13**	2	2	297	1579	1023	855
100	**9**	6	5	1	253	1076	255	1250
500	**4**	2	2	0	226	745	253	-
BFS	0	0	0	0	-	-	-	-

DriverLog								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	0	**8**	0	3	-	6801	-	4329
10	5	**12**	2	7	1227	1439	1061	435
50	0	**12**	1	1	-	541	129	136
100	0	**11**	0	1	-	275	-	98
500	0	**1**	0	0	-	94	-	-
BFS	1	1	0	**2**	154	138	-	332

Depots								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	**4**	3	2	2	1526	790	588	588
10	5	6	**7**	6	3259	3295	2042	715
50	2	6	**7**	3	517	467	707	392
100	4	**7**	6	5	43	207	147	54
500	6	**11**	**11**	5	47	53	53	38
BFS	**4**	2	2	2	106	48	48	48

FreeCell								
	Problems solved				Median plan length			
b	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$	LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
1	7	**9**	5	5	132	123	125	133
10	**23**	21	**23**	19	89	89	85	71
50	**25**	19	24	24	69	66	68	68
100	24	21	22	**28**	68	65	65	72
500	**19**	4	21	**19**	61	55	62	61
BFS	23	20	**27**	25	104	97	104	104

Figure 13: Experimental results for various learning beam widths. For each domain, we show the number of solved problems and the median plan length of the solved problems. A dash in the table indicates that the planner was unable to solve any of the problems.

previous results. This included an analysis of convergence given ambiguous training data, giving a result that highlights the trade-off between the amount of allowed search and the difficulty of the resulting learning problem. Our experiments in the domain of automated planning showed that the approach has benefits compared to existing learning and non-learning state-space search planners. These results complement the positive empirical results in structured classification (Daumé III and Marcu, 2005) showing the general utility of the method.

In future work, we plan to extend the algorithms described here to allow for feature induction and more robust parameter estimation. We are also interested in studying learning in the context of search for other search strategies such as best-first and k-best-first search. In our initial investigations, we have found that the LaSO-style approach for these strategies has great difficulty in automated planning due to the very large depths of the search spaces, which makes it difficult to "assign credit" to search errors. This suggests that a key aspect of future work is to understand general credit-assignment mechanisms in the context of error-driven learning for search. Another important direction is to consider the application of these methods to new problem domains, in particular we are interested in more complex planning domains that include concurrency, durative actions, and uncertainty. It will also be interesting to consider learning beam-search heuristics for other search-based formulations of planning such as partial-order planning where the search is conducted directly in the space of partial-order plans.

Acknowledgments

Some of the material in this paper was first published at ICML-2007 (Xu and Fern, 2007) and IJCAI-07 (Xu et al., 2007).

Appendix A. Relation to Structured Classification

This Appendix assumes that the reader is familiar with the material in Section 3. The learning framework introduced in Section 2.2 is motivated by automated planning, with the objective of finding a goal node. It is important to note that the learning objective does not place a constraint on the rank of a goal node in the final beam compared to non-goal nodes, but rather only requires that there exists some goal node in the final beam. This is a natural formulation for automated planning where when solving test problems it is easy to test each beam to determine whether a goal node has been uncovered and to return a solution trajectory if one has. Thus, the exact ordering of the goal node in the final beam is not important with respect to finding solutions to planning problems.

In contrast, as described in the example at the end of Section 2.2, the formulation of structured classification as a search problem appears to require that we do pay attention to the rank of the goal nodes in the final beam. In particular, the formulation of Daumé III and Marcu (2005) requires the goal node to not only be contained in the final beam, but to be ranked higher than any other terminal node in the beam.

Since our formulation of the beam-search learning problem does not constrain the ranking of goal nodes relative to other nodes, it is not immediately clear how our formulation relates to structured classification. It turns out that these two formulations are polynomially equivalent, meaning that there is a polynomial reduction from each problem to the other. Thus, it is possible to compile away the explicit requirement that goal nodes have the highest rank in the final beam.

Below we adapt the definitions of the learning problems in Section 2.2 for structured classification. First, we introduce the notion of terminal node, which can be thought of as a possible solution to be returned by a structured classification algorithm, for example, a full parse tree for a sentence. We will denote the set of all terminal nodes as $\mathcal{T}$ and will assume a polynomial time test for determining whether a node is in this set. Note that some terminal nodes correspond to target solutions and others do not. When using beam search for structured classification the search is halted whenever a terminal node becomes highest ranked in the beam and the path leading to that terminal node is returned as the solution. Thus, successful learning must ensure both that no non-target terminal node ever becomes ranked first in any beam and also that eventually a target terminal node does become ranked first. This motivation leads to the following definitions for the breadth-first and best-first structured classification problems. Below, given the context of a weight vector w, we will denote the highest ranked node relative to w in a beam B by $B^{(1)}$.

Definition 17 (Breadth-First Structured Classification) *Given the input $\langle\{\langle S_i, P_i\rangle\}, b\rangle$, where b is a positive integer and $P_i = (P_{i,0}, \ldots, P_{i,d})$, the breadth-first structured classification problem asks us to decide whether there is a weight vector w such that for each S_i, the corresponding beam trajectory $(B_{i,0}, \ldots, B_{i,d})$, produced using w with a beam width of b, satisfies $B_{i,j} \cap P_{i,j} \neq \emptyset$ for each j, $B_{i,d}^{(1)} \in P_{i,d}$, and $B_{i,j}^{(1)} \notin \mathcal{T}$ for $j < d$?*

Definition 18 (Best-First Structured Classification) *Given the input $\langle\{\langle S_i, P_i\rangle\}, b\rangle$, where b is a positive integer and $P_i = (P_{i,0}, \ldots, P_{i,d})$, the best-first structured classification problem asks us to decide whether there is a weight vector w that produces for each S_i a beam trajectory $(B_{i,0}, \ldots, B_{i,k})$ of beam width b, such that $k \leq h$, each $B_{i,j}$ for $j < k$ contains at least one node in $\bigcup_j P_{i,j}$, $B_{i,k}^{(1)} \in P_{i,d}$, and $B_{i,j}^{(1)} \notin \mathcal{T}$ for $j < k$?*

We prove that these problems are polynomially equivalent to breadth-first and best-first consistency by showing that they are NP-complete. Since Section 3 proves that the consistency problems are also NP-complete we immediately get equivalence.

Theorem 19 *Breadth-first structured classification is NP-complete.*

Proof We can prove that the problem is in NP, following the structure of the proof of Theorem 4. Each certificate corresponds to a set of beam trajectories and has a size that is polynomial in the problem size. The certificate can be checked in polynomial time to see if for each i, it satisfies the conditions defined in Definition 17. From Lemma 3 in Section 3 we can then use the algorithm *TestTrajectories* in Figure 4 to decide whether there is a weight vector that generates the certificate in polynomial time. To show hardness we reduce from breadth-first consistency for the class of problems where $b = 1$, $d = 1$, $c = 6$, $t = 3$, and $n \geq 1$, which from Figure 6 is NP-complete. Since for this class the search spaces have depth 1 and the beam width is 1 it is easy to see that for any problem in this class, a weight vector is a solution to the consistency problem if and only if it is a solution to the structured classification problem. This shows that breadth-first structured classification is NP-hard and thus NP-complete. ■

Using an almost identical proof we can prove the same result for best-first structured classification.

Theorem 20 *Best-first structured classification is NP-complete.*

References

Shivani Agarwal and Dan Roth. Learnability of bipartite ranking functions. In *Proceedings of the Conference on Learning Theory*, 2005.

Ricardo Aler, Daniel Borrajo, and Pedro Isasi. Using genetic programming to learn and improve control knowledge. *Artificial Intelligence*, 141(1-2):29–56, 2002.

José Luis Ambite, Craig A. Knoblock, and Steven Minton. Learning plan rewriting rules. In *Proceeding of Artificial Intelligence Planning Systems*, pages 3–12, 2000.

Blai Bonet and Hećtor Geffner. Planning as heuristic search: New results. In *Proceedings of the European Conference on Planning*, pages 360–372, 1999.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with the perceptron algorithm. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2002.

Hal Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the International Conference on Machine Learning*, 2005.

Tara A. Estlin and Rymond J. Mooney. Multi-strategy learning of search control for partial-order planning. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 843–848, 1996.

Alan Fern, Sungwook Yoon, and Robert Givan. Approximate policy iteration with a policy language bias: Solving relational markov decision processes. *Journal of Artificial Intelligence Research*, 25:85–118, 2006.

Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence Journal*, 3(1–3):251–288, 1972.

Michael R. Garey and David S. Johnson, editors. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, New York, 1979.

Klaus-Uwe Hoffgen, Hans-Ulrich Simon, and Kevin S. Van Horn. Robust trainability of single neurons. *Journal of Computer and System Sciences*, 50(1):114–125, 1995.

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

Yi-Cheng Huang, Bart Selman, and Henry Kautz. Learning declarative control rules for constraint-based planning. In *Proceedings of Seventeenth International Conference on Machine Learning*, pages 415–422, 2000.

Rong Jin and Zoubin Ghahramani. Learning with multiple labels. In *Proceedings of the Sixteenth Annual Conference on Neural Information Processing Systems*, 2002.

Leonid G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20(1):191–194, 1979.

Roni Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113(1-2): 125–148, 1999.

Tomás La Rosa, Angel García Olaya, and Daniel Borrajo. Using cases utility for heuristic planning improvement. In *Proceedings of the Seventh International Conference on Case Based Reasoning*, 2007.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289, 2001.

Mario Martin and Hector Geffner. Learning generalized policies in planning domains using concept languages. In *Proceedings of Seventh International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

Drew McDermott. PDDL- the planning domain definition language. In *The 1st International Planning Competition*, 1998.

Steven Minton. Quantitative results concerning the utility of explanation-based learning. In *Proceedings of National Conference on Artificial Intelligence*, 1988.

Steven Minton, editor. *Machine Learning Methods for Planning*. Morgan Kaufmann Publishers, 1993.

Steven Minton, Jaime G. Carbonell, Craig A. Knoblock, Daniel Kuokka, Oren Etzioni, and Yolanda Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence*, 40: 63–118, 1989.

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

Albert B. Novikoff. On convergence proofs on perceptrons. In *Symposium on the Mathematical Theory of Automata*, pages 615–622, 1962.

Frank Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

John Slaney and Sylvie Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125:119–153, 2001.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In *Neural Information Processing Systems Conference*, 2003.

Manuela M. Veloso, M. Alicia Pérez, and Jamie G. Carbonell. Nonlinear planning with parallel resource allocation. In *Workshop on Innovative Approaches to Planning, Scheduling and Control*, pages 207–212, 1991.

Yuehua Xu and Alan Fern. On learning linear ranking functions for beam search. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2007.

Yuehua Xu, Alan Fern, and Sungwook Yoon. Discriminative learning of beam-search heuristics for planning. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2007.

Sungwook Yoon, Alan Fern, and Robert Givan. Inductive policy selection for first-order MDPs. In *Proceedings of Eighteenth Conference in Uncertainty in Artificial Intelligence*, 2002.

Sungwook Yoon, Alan Fern, and Robert Givan. Learning heuristic functions from relaxed plans. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 2006.

Sungwook Yoon, Alan Fern, and Robert Givan. Learning control knowledge for forward search planning. *Journal of Machine Learning Research*, 9:683–718, 2008.

Terry Zimmerman and Subbarao Kambhampati. Learning-assisted automated planning: Looking back, taking stock, going forward. *AI Magazine*, 24(2)(2):73–96, 2003.