AN ABSTRACT OF THE DISSERTATION OF

Yuehua Xu for the degree of Doctor of Philosophy in Computer Science presented on August 4, 2010.

Title: Learning Ranking Functions for Efficient Search

Abstract approved: ______________________________

Alan Fern

This dissertation explores algorithms for learning ranking functions to efficiently solve search problems, with application to automated planning. Specifically, we consider the frameworks of beam search, greedy search, and randomized search, which all aim to maintain tractability at the cost of not guaranteeing completeness nor optimality. Our learning objective for each of these frameworks is to induce a linear ranking function for guiding the search that performs nearly as well as unconstrained search, hence gaining computational efficiency without seriously sacrificing optimality.

We first investigate the problem of learning ranking functions to guide beam search, with a focus on learning feature weights given a set of features. We present a theoretical analysis of the problem's computational complexity that identifies the core efficient and hard subclasses. In addition we study online learning algorithms for the problem and analyze their convergence properties.

The algorithms are applied to automated planning, showing 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.

Next, we study the problem of automatically learning both features and weights to guide greedy search. We present a new iterative learning algorithm based on RankBoost, an efficient boosting algorithm for ranking and demonstrate strong empirical results in the domain of automated planning.

Finally, we consider the problem of learning randomized policies for guiding randomized greedy search with restarts. We pose this problem in the framework of reinforcement learning and investigate policy-gradient algorithms for learning both features and weights. The results show that in a number of domains this approach is significantly better than those obtained for deterministic greedy search.

Learning Ranking Functions for Efficient Search

by

Yuehua Xu

A DISSERTATION

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Presented August 4, 2010
Commencement June 2011

ACKNOWLEDGEMENTS

I am profoundly grateful to my advisor, Alan Fern. I was very fortunate to have Dr. Fern as my mentor throughout the time it took me to complete my PhD program. Dr. Fern is incredibly knowledgeable, energetic, inspirational and supportive. His commitment to the pursuit of excellence in research work guided me through the past six years, and I am sure it will benefit me a lot in my future career.

I would like to thank my committee members, Tom Dietterich, Prasad Tadepalli, Raviv Raich, and Jon Herlocker. They generously contributed their time and expertise to better shape my research work. I also thank Karen Dixon for taking her time off to serve as Graduate Council Representative in my committee. I am immensely grateful for having been given the opportunity of spending these years with my fellow graduate students, faculty, and staff here at Oregon State University.

I want to especially thank Sungwook Yoon for providing me with the planners and feature learner. Without his help, I wouldn't be able to build my research work. I am also grateful to Horst Samulowitz, my internship mentor at Microsoft Research Cambridge, for giving me the opportunity to work on the satisfiability problems.

Last in deed but first in thought, I want to express my deepest gratitude to my family. Without the tremendous support and encouragement from them, I couldn't have made it this far. This dissertation is dedicated to them.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

DEDICATION

To my family.

Chapter 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. There are a number of strategies that attempt to maintain tractability of heuristic search such as beam search, greedy search, or randomized search. One common property of these search strategies is that they all prune away most nodes in the search queue. Due to this pruning, these search strategies are not guaranteed to be complete nor optimal. However, if the heuristic is good enough to guide the search process along a good solution path, then the solution will be found quickly.

This dissertation will investigate the problem of learning heuristics, or ranking functions, that allow these search strategies 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 training instance 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 find a ranking function that can efficiently guide the search process to find at least one target path. Such a ranking function allows the corresponding search strategy to efficiently solve all of the training instances, and ideally new search problems for which the training set is representative.

In order to learn the ranking function, we first assume that a set of features is provided for the given training set. Under this assumption, the ranking function can be represented by a weight vector w and consequently our learning task becomes to select w. This problem has been considered in the context of structured classification [Daumé III and Marcu, 2005], and a perceptron-style algorithm, known as *learning as search optimization (LaSO)*, is proposed to solve it. Motivated by their success, we formally define the learning problem for finding a weight vector that guides beam search to solve all training instances. One key contribution of this dissertation is to analyze the weight learning problem theoretically, in terms of its computational complexity and the convergence properties of various learning algorithms. Also, we provide an empirical evaluation of these weight learning algorithms, in the context of automated planning.

While showing good empirical results in automated planning, the above work requires features to be provided before learning. As a result, the performance of the learned ranking function is limited to the given features. In this dissertation, we also attempt to learn features automatically for a target planning domain. Here we focus on greedy search and consider the features in the form of action-selections rules, which are usually learned to define reactive policies in planning domains [Khardon, 1999; Martin and Geffner, 2000; Yoon *et al.*, 2002]. The learned ranking function, represented as a set of weighted action-selection rules, will assign numeric scores to potential state transitions. These scores can then be used to guide greedy search for solving the planning problems. This approach allows for information from multiple rules to be combined to help maintain robustness to errors. Our learning approach is based on a combination of a heuristic rule learner and RankBoost [Freund *et al.*, 2003], an efficient boosting-style algorithm for learning ranking functions. The empirical results have shown significant promise in a number of planning domains.

One way to extend greedy search is to generate a randomized policy based on the ranking function. Given any search node v, we can normalize the numeric scores assigned by the ranking function on all children nodes of v, resulting in a probability distribution. Note that the search node with higher rank will have a higher probability. The randomized policy then randomly selects a node according to the distribution. Here our learning objective is to learn a randomized policy that can find a solution path with only a small number of restarts. We formulate the learning problem in the framework of reinforcement learning, with the objective to maximize the expected average reward of the randomized policy. We then apply recent gradient-based reinforcement learning techniques [Baxter and Bartlett, 2000; Kersting and Driessens, 2008] for learning the features and optimizing the weights. Our empirical results have shown that by maximizing the expected average reward, we have learned randomized polices that work well in most planning domains.

1.1 Outline

This dissertation is organized as follows. Section 2 introduces the basic search concepts and Section 3 gives the background of our application domain - automated planning. Section 4 formulates the weight learning problem for beam search and studies its theoretical properties. Section 5 applies weight learning algorithm to automated planning and presents the experimental results. Section 6 gives our iterative learning algorithm for automatically inducing rule-based features to guide greedy search. Section 7 investigates the problem of learning randomizied polices. Finally, Section 8 concludes the dissertation with proposal for future research directions.

Chapter 2 – Basic Search Concepts

In this chapter, we formally define search spaces and describe the search paradigms used in the dissertation.

2.1 Search Space

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 dissertation, 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'.

2.2 Efficient Search

First we introduce two different beam search paradigms: breadth-first beam search and best-first beam search. 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;
- $C_{j+1} = \textbf{BreadthExpand}(B_j, s(\cdot)) = \bigcup_{v \in B_j} s(v)$ 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 $\textbf{BestExpand}(B_j, s(\cdot)) = B_j \cup s(v^*) - v^*$, where v^* is the unique highest ranked 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.

Greedy search is a special case of beam search with $b = 1$, which keeps only the highest ranked node at each search step. More formally, given a search space $S = \langle I, s(\cdot), f(\cdot), < \rangle$ and a weight vector w, greedy search generates a unique node sequence $(v_0, v_1, \ldots)$ where $v_0 = I$ is the initial node and v_i is the highest ranked node in $s(v_{i-1})$ according to r.

When the ranking function is not good enough, greedy search often leads to failures. While beam search can improve greedy search by having larger beam widths and explore more in the search space, another way of extending it is to randomize the search process. At each search step i, instead of selecting the highest ranked node, we consider randomly selecting v_i from $s(v_{i-1})$ according to a probability distribution. *Randomized Greedy Search* is a search process that adds randomization as above to greedy search and allows quick restarts. More details will be provided in Chapter 7.

Chapter 3 – Automated Planning Background

In this chapter, we first give background related to the application of automated planning. We then discuss the problem of learning to plan, with a brief summary of prior work in this area.

3.1 Automated Planning

Planning is a subfield of artificial intelligence that studies algorithms for selecting sequences of actions in order to achieve goals. In this dissertation, 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 $\mathrm{Pre}(a)$, and 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 $\mathrm{Pre}(a) \subseteq \omega$. The application of an (applicable) action a to ω results in the new state $\omega' = (\omega \setminus \text{Del}(a)) \cup \text{Add}(a)$. 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 $(a_1, \ldots, a_l)$, where the application of the sequence starting in state ω leads to a goal state ω' where $g \subseteq \omega'$. In this dissertation, 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 3.1 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

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

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

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

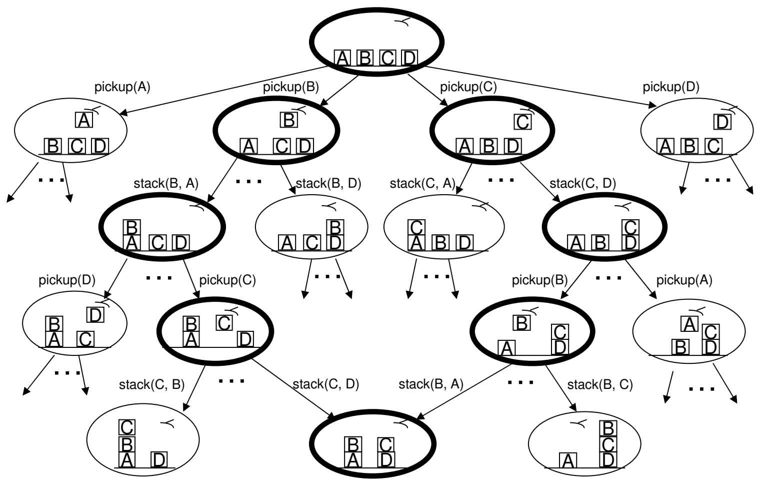

Figure 3.1: An example from automated planning.

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 $g = \{on(C, D), on(B, A)\}$, then one solution for the problem, as shown in Figure 3.1, is the action sequence ($pickup(B)$, $stack(B, A)$, $pickup(C)$, $stack(C, D)$).

There has been much recent progress in automated planning. One of the most successful approaches, and the one most relevant to our work, 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].

3.2 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 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 efficient search algorithms. Given a representative training set of problems from a planning domain, our approach first solves the problems using potentially expensive search (e.g., complete search), guided by an existing heuristic. These solutions are then used to learn a ranking function that can guide a small width beam search or greedy search to the same solutions. The hope is that the learned ranking function will then generalize and allow for the quick solution of new problems that could not be practically solved before learning.

3.3 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 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, which 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 obtain 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 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, Fern and Givan 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 [Yoon *et al.*, 2006]. 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 dissertation is to develop a learning mechanism that is more tightly integrated with the search process. 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.

Chapter 4 – Learning Weights for Linear Ranking Functions

In this chapter, we assume features are provided and investigate the problem of learning weights for beam search. First, we formally define the weight learning problems for breadth-first beam search and best-first beam search respectively. Next, we study their theoretical properties, in terms of computational complexity and convergence of various learning algorithms. The work presented in this chapter has been published in [Xu and Fern, 2007; Xu *et al.*, 2009a].

4.1 Weight 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.

Example from Automated Planning. Figure 3.1, 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.

Example from Structured Classification. Daumé III and Marcu considered learning ranking functions to control beam search in the context of structured classification [Daumé III and Marcu, 2005]. 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 4.1 shows how Daumé III and Marcu

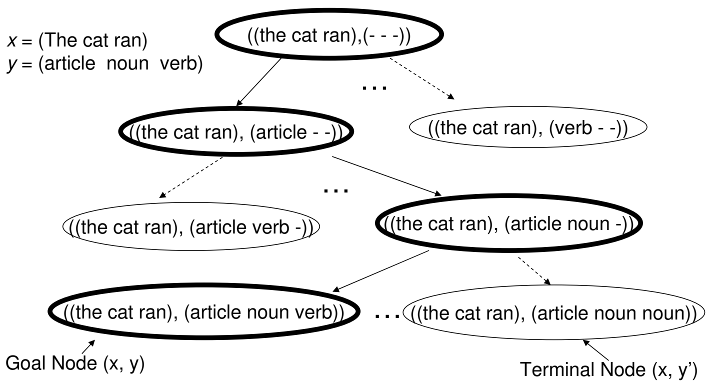
x = (The cat ran)
y = (article noun verb)
((the cat ran),(- - -))
((the cat ran), (article - -))
((the cat ran), (verb - -))
((the cat ran), (article verb -))
((the cat ran), (article noun -))
((the cat ran), (article noun verb))
((the cat ran), (article noun noun))
Goal Node (x, y)
Terminal Node (x,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

Figure 4.1: An example from structured classification.

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 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 [Daumé III and Marcu, 2005]. 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 example in Figure 4.1, 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 the Appendix. Thus, all of the results in this paper apply equally well to the structured-classification formulation of [Daumé III and Marcu, 2005].

4.2 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 4.1, and b - the beam width. Figure 4.2 gives a pictorial depiction of these key problem parameters. 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.

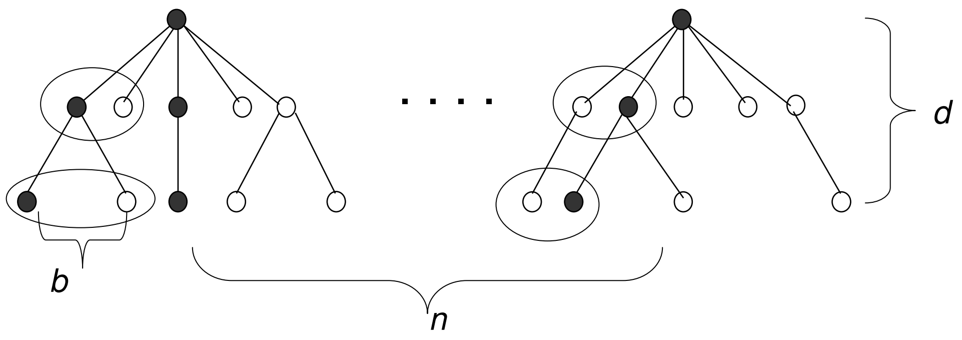

Figure 4.2: The key problem parameters: n - the number of training instances, d - the depth of target solution paths, b - the beam width. Not depicted in the figure are: c - maximum number of children of any node, t - the maximum target width of any example.

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.

4.2.1 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 4.3 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 1 *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 4.3) 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 $ndcb^2$

```
**ExhaustiveAlgorithm** $(\{\langle S_i, P_i\rangle\}, b)$
$\Gamma$ = **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 4.3: The exhaustive algorithm for breadth-first consistency.

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 1 *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 1 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 4.3. 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 2 *The procedure ExhaustiveAlgorithm (Figure 4.3) 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 $\text{poly}(m, ndcb^2)$. The total time complexity then is $O\left((tbdn + \text{poly}(m, ndcb^2))\,(cb)^{bdn}\right) = O\left((t + \text{poly}(m))\,(cb)^{bdn}\right)$.

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.

Theorem 3 *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 1 we can check for a solution weight vector for these trajectories in polynomial time.

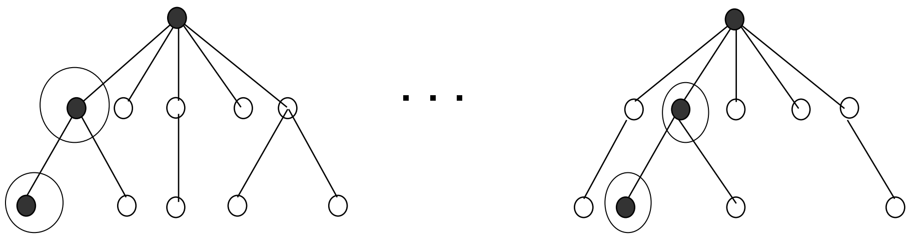

Figure 4.4: A tractable class of breadth-first consistency, where $b = 1$ and $t = 1$.

This problem class, as depicted in Figure 4.4, 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].

4.2.2 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, e.g. 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 4 *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 5 *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 is 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, e.g. $q_{i'j'} = \neg u_k$, 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 6 *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 $y = \langle\langle \bar{S}_1, \bar{P}_1\rangle, b\rangle$ where $\bar{S}_1 = \langle I_1, \bar{s}(\cdot), \bar{f}(\cdot), \bar{<}\rangle$, and $\bar{P}_1 = (\{I_1\}, P_{1,1}, P_{2,1}, \ldots, P_{t,1})$. We define $\bar{s}(\cdot), \bar{f}(\cdot), \bar{<}$ as below.

- $\bar{s}(I_1) = s_1(I_1)$, $\bar{f}(I_1) = f_1(I_1)$;
- for each $i = 1, \ldots, n-1$

  $\forall v \in s_i(I_i)$, $\bar{f}(v) = f_i(v)$ and $\bar{s}(v) = s_{i+1}(I_{i+1})$;

  $\forall (v, v') \in s_i(I_i)$, $\bar{<}(v, v') =<_i (v, v')$;
- $\forall v \in s_n(I_n)$, $\bar{f}(v) = f_n(v)$;

  $\forall (v, v') \in s_n(I_n)$, $\bar{<}(v, v') =<_n (v, v')$.

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.

Table 4.1: Complexity results for breadth-first consistency.

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

Table 4.1 summarizes our main complexity results from this section for breadth-first consistency. Here K indicates a constant value and "poly" indicates that the quantity must be polynomially related to the problem size. Each row of Table

4.1 corresponds to a sub-class of the problem and indicates the computational complexity.

For best-first beam search, most of these results in Table 4.1 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.

4.3 Convergence of Online Updates

In section 4.2, 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.

4.3.1 Online Perceptron Updates

Figure 4.5 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 $P_{i,j}$ is 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 the 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.

```
**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 4.5: The LaSO-BR online algorithm for breadth-first beam search.

Figure 4.6 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 did not experience this difficulty due to the bounded-depth nature of their search spaces [Daumé III and Marcu, 2005].

4.3.2 Previous Result and Counter Example

Adjusting to our terminology, Daumé III and Marcu defined a training set to be *linear separable* iff there is a weight vector that solves the corresponding consistency problem [Daumé III and Marcu, 2005]. For linearly separable data they also 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 3 (Search Margin) *The search margin of a weight vector w for a linearly separable training set is defined as $\gamma = min_{\{(v^*,v)\}}(w \cdot f(v^*) - w \cdot f(v))$,*

```
**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**
      $w \leftarrow w + \alpha \cdot \left( \frac{\sum_{v^* \in \bar{P} \cap C} f(v^*)}{|\bar{P} \cap C|} - \frac{\sum_{v \in B} f(v)}{b} \right)$
      $B \leftarrow \bar{P} \cap C$
**return**
```

Figure 4.6: The LaSO-BST online algorithm for best-first beam search.

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 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 [Daumé III and Marcu, 2005]. 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 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 4.7, preference ordering*

$C < B < F < E < D < H < G$, and single target path $P = (\{A\}, \{B\}, \{E\})$.

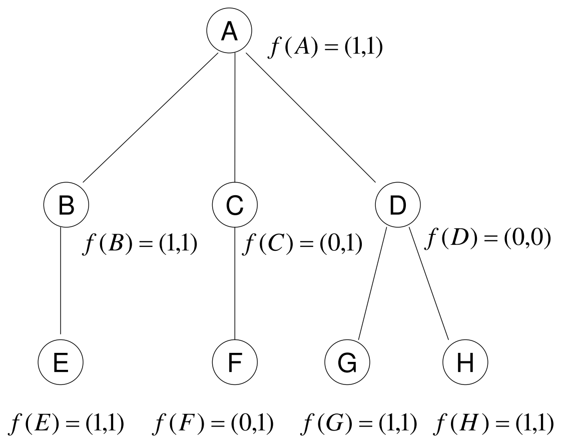

Figure 4.7: Counter example for convergence under positive search margin.

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,

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

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.

4.3.3 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 4 (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 7 *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 $w \cdot w^{k+1}$.

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

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 4.7 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 4.8, single target path $P = (\{A\}, \{B\}, \{E\})$, and preference ordering $C < B < E < F < G < D$.*

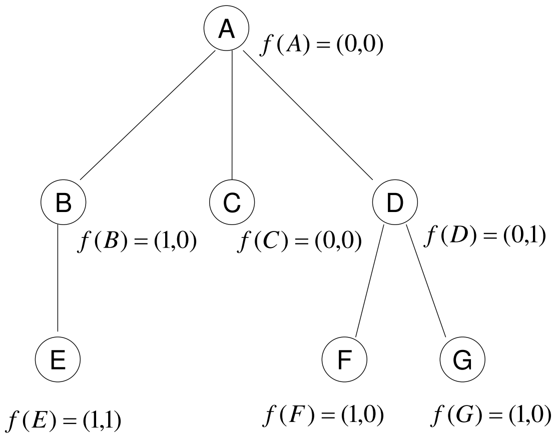

Figure 4.8: Counter example to convergence under positive level margin.

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.

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 5 (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 in [Daumé III and Marcu, 2005] and our own automated experiments both involve tree structured spaces.

Theorem 8 *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 7 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, we have introduced three different notions of margin: search margin, level margin, and global margin. Both LaSO-BR and LaSO-BST 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 implies 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.

4.3.4 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 6 (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 9 *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 7.

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 $|B'| < b'$.

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

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 7, 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.

Chapter 5 – Application of Weight Learning in Automated Planning

While the LaSO framework has been empirically evaluated in structured classification with impressive results [Daumé III and Marcu, 2005], its utility in other types of search problems has not been demonstrated. This chapter considers applying the above LaSO-style algorithms to automated planning, which also appears in our prior work [Xu *et al.*, 2007; 2009a].

The background related to automated planning has been given in Chapter 3. Roughly speaking, the planning problems 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. This is different to structured classification, which is most naturally viewed as an optimization problem, where we must search for a structured object that optimizes an objective function. 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.

5.1 Experimental Setup

We present experiments in eight STRIPS domains: Blocksworld, Depots, DriverLog, FreeCell, Pipesworld, Pipesworld-with-tankage, PSR and Philosopher. 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]

5.1.1 Domain Problem Sets

For each domain we needed to create a set of training problems and testing problems on which the learned ranking functions 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,

[1]The results in [Yoon *et al.*, 2006] indicated that their linear regression learning method was effective in Optical Telegraph. 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.

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.

5.1.2 Search Space Definition

We now describe the mapping between the planning problem described in Section 3.1 and the general search spaces described in Chapter 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/rank 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 chapter are described 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.

5.1.3 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 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 4.3.4.

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 that 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.

5.1.4 Ranking Function Representation and Domain Features

We consider learning ranking functions that are represented as weighted linear combinations of features, that is, $H(v) = \Sigma_i w_i f_i(v)$ 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.*, 2008a] 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, i.e. 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 equal 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 3.1, $f_1(v_0) = f_1((\omega_0, g)) = 4$ 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, i.e. 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 3.1 $f_2(v_0) = 2$ and $f_2(v_1) = 2$.

Since our work in this dissertation is focused on learning ranking functions, we refer to [Yoon *et al.*, 2008a] 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.*, 2008a] and tune their weights. In our experiments, this prior work gave us 15 features in Blocksworld, 3 features in Depot, 22 features in DriverLog, 3 features in FreeCell, 35 features in Pipesworld, 11 features in Pipesworld-with-tankage, 54 features in PSR and 19 features in Philosopher. In all cases, we include FF's relaxed-plan-length heuristic as an additional feature.

5.2 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. 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.

Training Time. Table 5.1 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 the number of features, the number of actions, the 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 only to 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.

Description of Tables. Before presenting our results, we will first provide an overview of the information contained in our results tables. Table 5.2 compares the performance of LaSO-BR$_{10}$ to three other algorithms,

- LEN : beam search using FF's relaxed plan length heuristic

Table 5.1: The average training time required by LaSO-BR per iteration for all training instances (seconds).

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

- 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 with 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.

In Table 5.2, the columns are 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 for each domain corresponding to using full best-first search (BFS) with an infinite beam width, which is the native search procedure used by FF. We first evaluate the performance of these planners by the number of solved problems. A planner solving more problems has better performance. When two planners solve the same number of problems, we break the tie by the median plan length of solved problems. The one with shorter plans is better. For example, the table shows that the ranking function learned via LaSO-BR$_{10}$ solves 26 Blocksworld testing problems with a median solution length of 139 using a testing beam width of 50, and solved 19 problems with a median solution length of 142 using BFS.

Table 5.3 is similar in structure to Table 5.2 but compares the performance of ranking functions learned using LaSO-BR with a variety of training beam widths and evaluated using a variety of testing beam widths. For example, the upper right-most data point gives the performance of LaSO-BR using a learning beam width of 100 and a testing beam width of 1.

Performance Across Testing Beam Width. From Table 5.2, 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 increases 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.

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 that the ranking function will be confused by “bad” states. This is also one possible reason why

Table 5.2: Experimental results for our weight learning algorithms. 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 median plan length is not available while none of the problems can be solved.

	b	Problem solved (Median plan length)			
		LEN	U	LR	LaSO-BR$_{10}$
Blocksworld	1	13(3318)	0(-)	11(938)	**24(499)**
	10	22(449)	0(-)	19(120)	**24(293)**
	50	20(228)	0(-)	19(64)	**26(139)**
	100	19(110)	0(-)	20(67)	**24(144)**
	500	17(80)	0(-)	**23(74)**	17(96)
	BFS	5(80)	0(-)	13(76)	**19(142)**
Pipesworld	1	11(114)	13(651)	8(2476)	**16(2853)**
	10	17(112)	17(360)	21(194)	**23(222)**
	50	18(34)	19(167)	21(89)	**26(80)**
	100	18(32)	16(39)	21(60)	**24(62)**
	500	21(30)	18(33)	21(31)	**25(53)**
	BFS	**15(44)**	7(54)	7(42)	15(54)
Pipesworld-with-tankage	1	6(119)	4(416)	2(1678)	**7(291)**
	10	6(68)	8(603)	**9(399)**	8(117)
	50	6(61)	5(111)	6(94)	**11(122)**
	100	5(54)	4(105)	5(43)	**8(55)**
	500	5(42)	6(97)	4(41)	**10(76)**
	BFS	**5(59)**	3(60)	2(126)	3(100)
PSR	1	0(-)	0(-)	0(-)	0(-)
	10	1(516)	**20(157)**	13(151)	13(193)
	50	13(99)	**17(109)**	16(99)	10(97)
	100	13(103)	**15(89)**	13(89)	6(85)
	500	**4(55)**	4(59)	2(48)	1(39)
	BFS	13(89)	0(-)	**21(131)**	21(141)
Philosopher	1	0(-)	**33(363)**	**33(363)**	**33(363)**
	10	0(-)	**33(363)**	**33(363)**	11(1154)
	50	0(-)	6(215)	**23(308)**	13(1579)
	100	0(-)	16(292)	**18(281)**	6(1076)
	500	0(-)	**7(220)**	**7(220)**	2(745)
	BFS	0(-)	**33(363)**	**33(363)**	0(-)
Depots	1	1(462)	1(790)	2(411)	**3(790)**
	10	4(195)	1(28)	4(981)	**6(3295)**
	50	3(25)	4(511)	5(51)	**6(467)**
	100	4(232)	**7(157)**	3(26)	7(207)
	500	5(38)	4(62)	6(39)	**11(53)**
	BFS	2(46)	2(48)	**3(33)**	2(48)
DriverLog	1	0(-)	0(-)	0(-)	**8(6801)**
	10	3(789)	0(-)	0(-)	**12(1439)**
	50	4(108)	8(177)	0(-)	**12(541)**
	100	1(98)	**11(147)**	0(-)	11(275)
	500	0(-)	**3(86)**	0(-)	1(94)
	BFS	**6(162)**	2(181)	0(-)	1(138)
FreeCell	1	5(96)	7(120)	4(146)	**9(123)**
	10	20(82)	**22(117)**	19(243)	21(89)
	50	23(65)	**24(73)**	12(102)	19(66)
	100	20(65)	18(63)	7(70)	**21(65)**
	500	3(53)	3(55)	2(59)	**4(55)**
	BFS	**23(78)**	20(87)	12(111)	20(97)

Table 5.3: 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 median plan length is not available while none of the problems can be solved.

	b	Problem solved (Median plan length)			
		LaSO-BR$_1$	LaSO-BR$_{10}$	LaSO-BR$_{50}$	LaSO-BR$_{100}$
Blocksworld	1	**27(840)**	24(499)	18(92)	13(314)
	10	**27(206)**	24(293)	20(96)	19(150)
	50	**27(180)**	26(139)	23(72)	24(82)
	100	**25(236)**	24(144)	23(72)	23(86)
	500	23(122)	17(96)	19(62)	**24(77)**
	BFS	**21(116)**	19(142)	18(73)	17(124)
Pipesworld	1	16(1803)	16(2853)	**21(1403)**	15(6958)
	10	**25(227)**	23(222)	23(179)	21(270)
	50	25(74)	**26(80)**	25(119)	22(75)
	100	**27(146)**	24(62)	23(104)	22(47)
	500	23(60)	**25(53)**	20(61)	21(37)
	BFS	14(59)	**15(54)**	13(103)	8(42)
Pipesworld-with-tankage	1	5(55)	**7(291)**	2(197)	7(300)
	10	8(103)	8(117)	8(68)	**10(77)**
	50	9(48)	**11(122)**	8(37)	9(42)
	100	8(53)	8(55)	10(122)	**10(55)**
	500	9(30)	**10(76)**	5(39)	10(96)
	BFS	**6(48)**	3(100)	4(70)	6(63)
PSR	1	0(-)	0(-)	0(-)	0(-)
	10	12(182)	13(193)	3(550)	**14(205)**
	50	6(75)	10(97)	16(126)	**17(129)**
	100	3(82)	6(85)	10(113)	**13(86)**
	500	2(61)	1(39)	**4(58)**	4(64)
	BFS	19(164)	21(141)	3(170)	**25(142)**
Philosopher	1	6(589(	**33(363)**	**33(363)**	0(-)
	10	**19(319)**	11(1154)	1(451)	1(1618)
	50	**13(297)**	13(1579)	2(1023)	2(855)
	100	**9(253)**	6(1076)	5(255)	1(1250)
	500	**4(226)**	2(745)	2(253)	0(-)
	BFS	0(-)	0(-)	0(-)	0(-)
DriverLog	1	0(-)	**8(6801)**	0(-)	3(4329)
	10	5(1227)	**12(1439)**	2(1061)	7(435)
	50	0(-)	**12(541)**	1(129)	1(136)
	100	0(-)	**11(275)**	0(-)	1(98)
	500	0(-)	**1(94)**	0(-)	0(-)
	BFS	1(154)	1(138)	0(-)	**2(332)**
Depots	1	**4(1526)**	3(790)	2(588)	2(588)
	10	5(3259)	6(3295)	**7(2042)**	6(715)
	50	2(517)	6(467)	**7(707)**	3(392)
	100	4(43)	**7(207)**	6(147)	5(54)
	500	6(47)	**11(53)**	**11(53)**	5(38)
	BFS	**4(106)**	2(48)	2(48)	2(48)
FreeCell	1	7(132)	**9(123)**	5(125)	5(133)
	10	23(89)	21(89)	**23(85)**	19(71)
	50	**25(69)**	19(66)	24(68)	24(68)
	100	24(68)	21(65)	22(65)	**28(72)**
	500	**19(61)**	4(55)	21(62)	**19(61)**
	BFS	23(104)	20(97)	**27(104)**	25(104)

performance tends to decrease with larger testing beam widths.

LaSO-BR$_{10}$ Versus No Learning. From Table 5.2, 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 ranking functions 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.

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 learned linear-regression heuristics are shown in the columns labeled LR in Table 5.2.

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 cases 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 to 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 while regression happens to work well.

Effects of Learning Beam Width. Table 5.3 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 all testing beam widths except 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 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 testing 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.

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

[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.

shown in the bottom row of each table in Table 5.2 and 5.3.

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.

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 improve 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.

Chapter 6 – Learning Features and Weights for Greedy Search

While the LaSO-BR algorithm has shown promising results in automated planning, it is limited to tuning the weights of previously selected features. In this chapter, we study the problem of automatically learning features and weights for guiding greedy search. The work presented in this chapter has been published in [Xu *et al.*, 2009b; 2010].

6.1 Introduction

It is often the case that search problems must be solved quickly in order for their solutions to be usefully applied. Such scenarios often arise due to real-time constraints, but also in problem-solving frameworks that solve complex problems by reduction to a number of simpler search problems, each of which must be solved quickly. For example, some approaches to probabilistic planning involve generating and solving many related deterministic planning problems [Yoon *et al.*, 2008b; Kolobov *et al.*, 2009]. Greedy search is one approach to finding solutions quickly by pruning away most nodes in a search space. We have investigated it as a special case of beam search in prior chapters. However, the LaSO-BR algorithm in Chapter 4 only learns the weights for a human-provided set of features and did not provide a feature learning mechanism. In this chapter, we consider learning weights and features, to guide greedy search in planning domains based on training solutions to example problems.

One prior approach to learning greedy control knowledge has been to learn action-selection rules for defining reactive policies tuned to a particular planning domain [Khardon, 1999; Martin and Geffner, 2000; Yoon *et al.*, 2002]. Given a good reactive policy, a planning problem from the corresponding domain can be quickly solved without search. While action-selection rules are an intuitively appealing form of control knowledge and give good results in certain domains, experience with existing learning algorithms has shown that in many domains the learned reactive policies are often imperfect and result in poor planning performance.

There are at least three possible reasons for this lack of robustness. **(1)** Policies have typically been defined as decision-lists of action-selection rules, which can be quite sensitive to variations in the training data. In particular, each decision made by a decision list is based on a single rule, rather than a vote among a number of rules, each providing its own evidence. **(2)** Learning from example plans (i.e. state-action sequences) leads to ambiguity in the training data. In particular, there are often many good actions in a state, yet the training data will generally only contain one of them arbitrarily. Attempting to learn a policy that selects the arbitrary training-data actions over other, inherently equal, actions can lead to extremely difficult learning problems. **(3)** Prior approaches to learning greedy policies from example plans typically do not consider the search performance of the learned policy. These approaches select a policy based on an analysis of the training data alone, but never actually observe the search performance of the selected policy.

In this chapter, we describe a new form of control knowledge and learning algorithm for addressing the above three shortcomings. **(1)** Recognizing that action-selection rules are able to capture useful knowledge, we attempt to learn and use such rules in a more robust way. In particular, we use sets of weighted rules to define a ranking function on state transitions, which allows multiple rules to vote at each decision point. We use a variant of the powerful RankBoost algorithm to learn both the rules and rule weights. **(2)** To deal with the problem of ambiguous training data, we derive partially-ordered plans from the original sequential training plans. These provide more accurate information about which actions are good and bad in each state. We then define our learning goal to be that of forcing greedy search to remain consistent with the partially-ordered plan, rather than the original action sequence. **(3)** We introduce a novel iterative learning algorithm that takes the search performance into account. On each iteration, the algorithm conducts a greedy search with the current knowledge and uses the observed search errors to guide the next round of learning. Thus, the learning is tightly integrated with the search process.

To our knowledge, there is only one other prior work that has integrated learning with search and that also attempts to learn features and weights [Daumé III *et al.*, 2009]. That work attempts to learn greedy policies as linear combinations of component policies (or features) to guide greedy search for structured-prediction problems. However, the work makes a number of assumptions that are often valid for structured prediction, but rarely valid for automated planning or similar combinatorial search problems. In particular, the work assumes the ability to compute the optimal policy at any state that might be generated on the training search problems, which is highly impractical for the planning problems we are interested in.

The remainder of the chapter is organized as follows. First, we introduce the form of our rule-based ranking function. Then, we present a new iterative algorithm for learning the ranking function, followed by a description of our rule learner. We finally present experimental results and conclude.

6.2 Rule-based Ranking Functions

We consider ranking functions of state transitions that are represented as linear combinations of features, where our features will correspond to action-selection rules. Following prior work [Yoon *et al.*, 2008a] that used taxonomic syntax to define action-selection rules for reactive policies, each of our action-selection rules has the form:

$$u(z_1, \ldots, z_k) : L_1, L_2, \ldots, L_m \tag{6.1}$$

where u is a k-argument action type and the z_i are argument variables. Each L_i here is a literal of the form $z \in E$ where $z \in \{z_1, \ldots, z_k\}$ and E is a taxonomic class expression. Given a state-goal pair (s, g), each class expression E represents a set of objects in a planning problem, so that each literal can be viewed as constraining a variable to take values from a particular set of objects. For example, **holding** is a taxonomic class expression in the Blocksworld domain and it represents the set of blocks that are being held in the current state. More complex class expressions can be built via operations such as intersection, negation, composition, etc. For example, the class expression **ontable** $\cap$ **gontable** represents the set of objects/blocks that are on the table in both the current state and goal. Given a state-goal pair (s, g) and a ground action a where $a = u(o_1, \ldots, o_k)$, the literal $z_j \in E$ is said to be true if and only if o_j is in the set of objects that is represented by E. We say that the rule *suggests* action a for state-goal pair (s, g) if all of the rule literals are true for a relative to (s, g).

Given a rule of the above form, we can define a corresponding feature function f on state transitions, where a state transition is simply a state-goal-action tuple (s, g, a) such that a is applicable in s. The value of $f(s, g, a) = 1$ iff the corresponding rule suggests a for (s, g) and otherwise $f(s, g, a) = 0$. An example rule in the Blocksworld domain is: **putdown**(x_1): $x_1 \in$ **holding** which defines a feature function f, where $f(s, g, a) = 1$ iff $a =$ **putdown**(o) and **holding**$(o) \in s$ for some object o, and is equal to zero for all other transitions.

Assume that we have a set of rules giving a corresponding set of feature functions $\{f_i\}$. The ranking function is then a linear combination of these rule-based features

$$F(s, g, a) = \sum_i w_i f_i(s, g, a)$$

where w_i is the corresponding real-valued weight of f_i. From this it is clear that the rank assigned to a transition is simply the sum of the weights of all rules that suggest that transition. In this way, rules that have positive weights can vote for transitions by increasing their rank, and rules with negative weights can vote against transitions by decreasing their rank.

6.3 Learning Weighted Rule Sets

In this section, we describe the traditional rank learning problem from machine learning and then formulate the learning-to-plan problem as a rank learning problem. Next we describe a variant of the RankBoost algorithm for solving the ranking learning problem. Finally, we describe our novel iterative learning algorithm based on RankBoost for learning weighted rule sets.

6.3.1 The Rank Learning Problem

Given a set of instances I, a ranking function is a function that maps I to the reals. We will view a ranking function as defining a preference ordering over I, with ties allowed. A rank learning problem provides us with training data that gives the relative rank between selected pairs of instances according to some unknown target partial ordering. The goal is to learn a ranking function that is (approximately) consistent with the target ordering across all of I based on the training data.

More formally a rank learning problem is a tuple (I, S), where I is a set of instances, and $S \subseteq I \times I$ is a training set of ordered pairs of instances. By convention, if $(v_1, v_2) \in S$ then the target is to rank v_1 higher than v_2. The learning objective is to learn a ranking function F that minimizes the number of misranked pairs of nodes relative to S [Freund *et al.*, 2003]. Here we say that F misranks a pair if $(v_1, v_2) \in S$ and $F(v_1) \leq F(v_2)$. The hope is that the learned ranking function will generalize so that it correctly ranks pairs outside of the training set.

It is typical to learn linear ranking functions of the form $F(v) = \sum_i w_i \cdot f_i(v)$, where the f_i are real-valued feature functions that assign scores to instances in I. The w_i are real-valued weights indicating the influence of each feature. In this chapter, the instances will correspond to possible state transitions from a planning domain and our features will be the rule-based features described above.

6.3.2 Learning-to-Plan as Rank Learning

We now describe two approaches for converting a learning-to-plan problem into a rank learning problem. Given a learning-to-plan training set $\{(x_i, y_i)\}$ for a planning domain, let s_{ij} be the j'-th state along the solution trajectory specified by y_i. Also let C_{ij} be the set of all candidate transitions out of s_{ij}, where a transition from s to s' via action a will be denoted by (s, a, s'). Finally, let $t_{ij} = (s_{ij}, a_{ij}, s_{i(j+1)})$ be the target transition out of s_{ij} specified by y_i. Our first conversion to a rank learning problem (I, S) defines the set of instances I to be the set of all possible state transitions in the planning domain. The set S is defined to require that t_{ij} be ranked higher than other transitions in C_{ij}, in particular, $S = \bigcup_{i,j}\{(t_{ij}, t) | t \in C_{ij}, t \neq t_{ij}\}$. Any ranking function that is consistent with S will allow for greedy search to produce all solutions in the training set and hence solve the learning-to-plan problem.

Unfortunately, for many planning domains, finding an accurate ranking function for the above rank learning problem will often be difficult or impossible. In particular, there are often many equally good solution trajectories for a planning problem other than those in the learning-to-plan training set, e.g. by exchanging the ordering of certain actions. In such cases, it becomes infeasible to require that the specific transitions observed in the training data be ranked higher than all other transitions in C_{ij} since many of those other transitions are equally good. To deal with this issue, our second conversion to rank learning attempts to determine which other transitions in C_{ij} are also good transitions. To do this we use the heuristic algorithm described in [Veloso *et al.*, 1991] to transform the given solution trajectories into partially ordered plans. The partially ordered plans contain the same set of actions as the totally ordered plan given by y_i but only include the necessary constraints on the action-ordering. Therefore, every partially ordered plan implicitly represents a set of solution paths, often an exponentially large set.

Given partially ordered plans for the examples in our learning-to-plan problem, we can now consider the learning goal of finding a ranking function such that greedy search will always remain consistent with the partially ordered plans. To do this we can generate a rank learning problem (I, S) that defines I as above, but defines the set S relative to the partial order plans as follows. Let $\delta(t, x_i)$ be a boolean function that determines whether a given transition $t = (s, a, s')$ is on the partially ordered plan for x_i. $\delta(t, x_i) = 1$ indicates that there exists a solution path consistent with the partially ordered plan that goes through t and $\delta(t, x_i) = 0$ otherwise. Given this we can arrive at an improved set of training pairs

$$S = \bigcup_{i,j} \{(t_1, t_2) \mid t_1, t_2 \in C_{ij}, \delta(t_1, x_i) = 1, \delta(t_2, x_i) = 0\}.$$

This definition of S specifies that for every state on the solution path corresponding to y_i, its outgoing transitions that are consistent with the partially ordered plan should be ranked higher than those that are not consistent. Intuitively, this learning problem will often be easier than the one defined earlier, since the learner is not forced to make arbitrary distinctions between equally good transitions (i.e. those consistent with the partially ordered plan).

Unfortunately, with this new form of training data, a subtle problem has been introduced. In particular, a ranking function that is consistent with the rank learning problem is no longer guaranteed to solve the training planning problems. That is, solving the rank learning problem does not necessarily solve the learning-to-plan problem. The reason for this is that the only transitions included in I are those that originate at nodes on the totally ordered training solutions (i.e. the union of transitions in the C_{ij}). However, a consistent ranking function might lead a greedy search to take a transition that leads off of the training solution, e.g. by selecting good/consistent transitions not in the totally ordered solution. The ranking function has not been trained on these transitions, and hence no guarantees can be made about its performance. One way to solve this problem would be to include all possible transitions in I and attempt to rank all transitions consistent with a partially ordered plan higher than all others. Unfortunately, there can be an exponentially large set of transitions consistent with a partially ordered plan, making this option intractable in general. In order to overcome this potential pitfall, we propose an iterative learning algorithm later in this section. Before that, we first describe how to solve a fixed ranking problem with a variant of the RankBoost algorithm.

6.3.3 RankBoost with Prior Knowledge

By converting our learning-to-plan problems to rank learning, we can now consider applying existing learning algorithms for ranking. RankBoost is a particularly effective algorithm that combines a set of weak learners in order to accurately rank a set of instances [Freund *et al.*, 2003]. Given a set of ordered pairs of instances S, RankBoost defines D to be a uniform distribution over all pairs in S. RankBoost's learning objective is to find a ranking function F that minimizes the rank loss with respect to D,

$$rLoss_D(F) = \sum_{(v_1,v_2)\in S} D(v_1, v_2) \cdot \psi(F(v_1) \leq F(v_2)),$$

where $\psi(\cdot)$ is 1 if its argument is true and 0 otherwise. This is equivalent to finding an F that minimizes the number of misranked pairs with respect to S.

RankBoost is an iterative algorithm that adds one feature to a linear ranking function on each iteration in order to improve the rank loss. To do this, on each iteration i it maintains a distribution D_i over all pairs in S, starting with D from above, which indicates the importance of each pair to be ranked correctly by the next learned feature. D_i is passed to the weak learner, which attempts to return a new feature f_i that achieves a good rank loss with respect to D_i. RankBoost then selects an appropriate weight w_i (details below) so that the resulting ranking function $F(v) = \sum_i w_i \cdot f_i(v)$ has a reduced rank loss over iteration $i - 1$. The distribution D_{i+1} then gets updated so that it decreases the emphasis on pairs ranked correctly by f_i and increases the emphasis on incorrectly ranked pairs. As a result, iteration $i + 1$ will concentrate more on pairs that have been misranked more often in previous iterations.

In our case, we consider a variant of RankBoost that takes into account prior knowledge provided by an initial ranking function. This is motivated by the fact that prior work [Yoon *et al.*, 2008a; Xu *et al.*, 2009a] has found it quite useful to incorporate state-of-the-art heuristics such as relaxed plan length [Hoffmann and Nebel, 2001] into the learned control knowledge. In addition, our overall algorithm (described later) will call RankBoost repeatedly, and it is beneficial to provide RankBoost with the best ranking function from previous calls as prior knowledge. Our variant algorithm takes any ranking function F_0 as input and learns weighted features that attempt to correct the mistakes of this ranking function.

As shown in Figure 6.1, the main idea is to modify the initial distribution according to the given ranking function F_0. The learned ranking function F is then equal to F_0 plus a linear combination of learned features that attempt to correct the mistakes of F_0. The following theorem proves a bound on the rank loss of F.

```
**RB-prior** $(S, F_0, k)$
// $S$ is the set of instance pairs.
// $F_0$ is the input ranking function.
// $k$ is the number of iterations.
**for** each pair $(v_1, v_2) \in S$
    $D_1(v_1, v_2) = \frac{\exp(F_0(v_2) - F_0(v_1))}{Z_0}$
**for** $i = 1, 2, \ldots, k$ :
    $f_i \leftarrow$ **Rule-Learner** $(S, D_i)$
    // Learning a ranking feature using distribution $D_i$
    Choose $w_i \in R$ // see text for our choice
    **for** each pair $(v_1, v_2) \in S$
        $D_{i+1}(v_1, v_2) = \frac{D_i(v_1,v_2) exp(w_i(f_i(v_2) - f_i(v_1)))}{Z_i}$
    where $Z_i$ is a normalization factor
**return** $F = F_0 + \sum_{i=1}^{k} w_i \cdot f_i$
```

Figure 6.1: The variant of RankBoost.

Theorem 10 *For any F_0, the rank loss on the training data of $F = F_0 + \sum_{i=1}^{k} w_i f_i$ returned by RB-prior satisfies rLoss$_D(F) \leq \prod_{i=0}^{k} Z_i$.*

The proof is a direct adaptation of the original RankBoost result [Freund *et al.*, 2003]. This bound indicates that if we can always maintain $Z_i < 1$, then the rank loss on the training data decreases exponentially fast. To achieve this, we follow the approach of [Freund *et al.*, 2003] for learning features and selecting weights, where a specialized formulation was presented for binary features, which is the case for our rule-based features. In particular, the weak learner attempts to find a feature f that maximizes $|r|$, where $r = \sum_{(v_1,v_2) \in S} D_i(v_1, v_2)(f(v_1) - f(v_2))$. The weight for the feature is set to $w_i = \frac{1}{2} \ln(\frac{1+r}{1-r})$. Provided that the weak learner can find a feature with $|r| > 0$, then it can be shown that $Z_i < 1$. Therefore, RB-prior is guaranteed to reduce the rank loss at an exponential rate provided that the weak learner can achieve a minimal guarantee of $|r| \geq \epsilon$ for any $\epsilon > 0$.

6.3.4 Iterative Learning Algorithm

As noted above, the conversions from learning-to-plan to rank learning included only a small fraction of all possible state transitions, in particular those transitions that originate from states in the training solution paths. For any transitions that are not included in S, the learning problem does not specify any constraints on their rank. Thus, when the learned ranking function leads greedy search to parts of the search space outside of S, the search is in unchartered territory and no guarantees can be given on the search performance of the learned weighted rule set on the training problems.

In order to help overcome this issue, we propose an iterative learning algorithm that integrates the above RB-prior algorithm with the search process. The goal is to form rank learning problems whose set of transitions in S are a better reflection of where a greedy search using the currently learned ranking function is going to go. In particular, it is desirable to include erroneous transitions resulting from greedy search with the current ranking function, where an erroneous transition is one that falls outside of the partially ordered plan of a training example. This allows for learning to focus on such errors and hopefully correct them.

More specifically, Figure 6.2 gives pseudo-code for our improved approach to learning ranking functions for planning. The top level procedure repeatedly con-

```
**IterativeLearning** $(\{x_i\}, \delta, F_0, k)$
// $x_i = (s_i, A, g_i)$ is a planning problem.
// $\delta$ is the function defined on partially ordered plans.
// $F_0$ is an initial ranking function, e.g. a planning heuristic.
// $k$ is the number of iterations of RB-prior for each generated ranking problem
$F \leftarrow F_0$          //initialize the ranking function
$S \leftarrow \emptyset$          // initialize the ranking problem
repeat until no improvement or a certain number of iterations
    $S \leftarrow S+$ ConstructRP $(\{x_i\}, \delta, F)$
    $F' \leftarrow$ RB-prior $(S, F, k)$
    $F \leftarrow$ LaSO-BR$_1(F')$
return $F$
```

```
ConstructRP $(\{x_i\}, \delta, F)$
$S \leftarrow \emptyset$
for each $x_i = (s_i, A, g_i)$
    $s \leftarrow s_i$
    repeat until $s \supseteq g_i$          // goal achieved
        $C \leftarrow$ all transitions $(s, a, s')$ out of $s$
        $C_+ \leftarrow \{t \mid t \in C \wedge \delta(t, x_i) = true\}$
        $C_- \leftarrow C - C_+$
        $S = S + \{(t, t') \mid t \in C_+, t' \in C_-\}$
        $s \leftarrow$ destination of highest ranked transition in $C_+$ according to $F$
return $S$
```

Figure 6.2: The iterative learning algorithm.

structs a ranking problem by calling **ConstructRP**, calls **RB-prior** to learn k new features on it, and then further optimizes the feature/rule weights by calling **LaSO-BR**$_1$ introduced in Chapter 4. In RB-prior, the weights are selected in order to minimize the rank loss. However, this does not always correspond exactly to the best weights for maximizing planning performance. Thus, to help improve the planning performance, we consider using the perceptron-style algorithm to further optimize the weights. This LaSO-BR$_1$ algorithm iteratively conducts greedy search and updates the weights in order to correct the search errors that have been made.

The key aspect of this iterative algorithm is that the training instances generated at each iteration depend on the performance of the currently learned ranking function. In particular, given the current ranking function F, the algorithm simulates the process of greedy search using F and then adds transition pairs to S along the greedy search path. At any node along the greedy search path all possible outgoing transitions are classified as being on or off the partial order plan via δ and pairs are added to S to specify that transitions on the partial order plan be ranked higher than those that are not. If, during this process, the greedy search should ever follow an erroneous transition according to δ, then the search will be artificially forced to follow the highest ranked good transition. This helps avoid adding pairs to S along paths that significantly diverge from the partially-ordered plans.

Convergence. Under certain assumptions the above iterative learning algorithm is guaranteed to converge to a ranking function that solves the training problems in a finite amount of time. This is a minimal property that a learning algorithm should have, but for most prior work on learning control knowledge, which does not take search performance into account, no such guarantees can be made.

The assumptions we make are as follows: 1) There exists a weighted rule set in our rule language which can correctly rank all nodes in the search space, according to δ defined on the partially ordered plans, 2) We have a weak rule learner which can always find a rule that achieves $|r| \geq \epsilon$ for some $\epsilon > 0$, which is a standard assumption in boosting theory, 3) Each call to RB-prior is run for enough iterations to achieve zero rank loss on its input ranking problem.

Under assumptions (2) and (3) we are guaranteed that each call to RB-prior will terminate in a finite amount of time with zero rank loss since the rank loss decreases exponentially fast as described earlier. Thus it remains to bound the number of calls to RB-prior. After each call to RB-prior, if the resulting ranking function does not solve a training problem then new training pairs of instances will be generated and added to the current set of training pairs S. Note that the training pairs are only generated for the transitions (s, a, s') out of s where s is a possible state generated by the corresponding partially ordered plan. Let m denote the number of possible states that can be reached by the partially ordered plans. The number of calls to RB-prior is then bounded by m. Unfortunately, m can be exponentially large in the size of the partially ordered plans. Unless further assumptions are made it is possible to construct example learning problems that match this worst case bound, showing the bound is tight.

In future work, we will investigate assumptions where convergence can be guaranteed in a small number of iterations. In particular, the worst case results are quite pathological since they assume that the learning algorithm never generalizes correctly to unseen states in the training data. It is likely that assumptions about generalization will allow for much tighter bounds.

6.3.5 Learning Action Selection Rules

The RankBoost algorithm assumes the existence of a weak learner that can be called to produce a ranking feature. In this section, we briefly introduce the rule learner we used. As shown in Figure 6.1, the input to the rule learner is the set of transition pairs S and a distribution D over S. In our case, each instance composing a pair in S is represented as a state-goal-action tuple (s, g, a), on which a rule-based feature can be evaluated. The learning objective is to find a rule that can maximize $|r|$ where $r = \sum_{(v_1,v_2)\in S} D(v_1, v_2)(f(v_1) - f(v_2))$.

For this purpose, we adapt the heuristic rule learner described in [Yoon *et al.*, 2008a] to find the best rule that can maximize $|r|$. Since the rule space is exponentially large, the approach performs a beam search over possible rules, where the starting rule has no literals in its body and each search step adds one literal to the body. The search terminates when the beam search is unable to improve $|r|$.

6.4 Experimental Results

We present experiments in seven STRIPS domains: Blocksworld, Depots, Driver-Log, FreeCell, Pipesworld, Pipesworld-with-tankage and Philosopher, which are already provided with details in Chapter 5. We also set a time cut-off of 30 CPU minutes and considered a problem to be unsolved if a solution was not found within the time cut-off. For our learning algorithms, the maximum number of learned rules is limited to 30. The learning algorithm will terminate when no improvement can be observed or the maximum number of rules has been reached. Note that the actual size of the learned rule set is usually smaller than 30 since the rule learner may output duplicated rules.

Description of Tables. Table 6.1 compares the performance of different approaches we used as well as algorithms in prior work for learning control knowledge for greedy search [Yoon *et al.*, 2008a; Xu *et al.*, 2009a]. These algorithms are:

- Yoon08 : three forms of control knowledge were learned in [Yoon *et al.*, 2008a] and were all evaluated for their ability to guide greedy search. The table entries labeled Yoon08 give the best performance among the three types of control knowledge as reported in that work. Results for Yoon08 are only given for our three IPC4 domains: Pipesworld, Pipesworld-with-tankage and Philosopher, for which our training and testing sets exactly correspond.

- RPL : greedy search with FF's relaxed plan length heuristic.

- LaSO-BR$_1$: greedy search with the ranking function learned by LaSO-BR$_1$, which is previously evaluated in Chapter 5.

- RB : greedy search with the weighted rule set that is learned by RB-prior when no prior knowledge is provided. Here the ranking problem is derived from just the states in the training trajectories. Learning is stopped when no improvement is observed or 30 rules are learned.

- RB-H : identical to RB except that we view the relaxed-plan length heuristic as prior knowledge and generate the ranking problem based on it.

- ITR : greedy search with the weighted rule set that is learned via our iterative learning approach starting with no prior knowledge. In this experiment, we do not accumulate data across iterations as described in Figure 6.2, for efficiency reasons, but rather pass only the most recent data to RB-prior. We choose to learn $k = 5$ rules for each ranking problem generated. Learning is terminated when no improvement is observed or 30 rules are learned in total.

- ITR-H: identical to ITR except that we use the relaxed-plan length heuristic as prior knowledge.

For an additional reference point, we also include the performance of FF [Hoffmann and Nebel, 2001] in Table 6.1 which is not constrained to perform greedy search. Each column of Table 6.1 corresponds to an algorithm and each row corresponds to a target planning domain. The planning performance is first evaluated with respect to the number of solved problems. When two algorithms solve the same number of problems, we will use the median plan length of the solved problems to break the tie.

Table 6.1: Experimental results for the learned ranking functions in greedy search. 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 median plan length is not available while none of the problems can be solved. N/A indicates that the result of the planner is not applicable here.

Problems solved (Median plan length)	FF	Yoon08	RPL	LaSO-BR$_1$
Blocksworld	10(77)	N/A	13 (3318)	27 (840)
Depots	14(63)	N/A	1 (462)	4 (1526)
DriverLog	3(119)	N/A	0 (-)	0 (-)
FreeCell	29(90)	N/A	5 (96)	7 (132)
Pipesworld	20(50)	0(-)	11 (114)	16 (1803)
Pipesworld-with-tankage	3(63)	0(-)	**6 (119)**	5 (55)
Philosopher	0(-)	0(-)	0 (-)	6 (589)

Problems solved (Median plan length)	RB	RB-H	ITR	ITR-H
Blocksworld	30 (126)	30 (166)	**30 (89)**	30 (118)
Depots	15 (661)	11 (129)	0 (-)	**23 (433)**
DriverLog	0 (-)	3 (2852)	0 (-)	**4 (544)**
FreeCell	5 (155)	7 (96)	2 (213)	**9 (92)**
Pipesworld	7 (1360)	17 (1063)	7 (1572)	**17 (579)**
Pipesworld-with-tankage	1 (1383)	6 (152)	3 (2005)	5 (206)
Philosopher	33 (875)	**33 (363)**	33 (875)	**33 (363)**

Table 6.3 and 6.2 provides more details of the approaches we used. We have a column "Learning iterations" indicating how many times the rule learner is called, i.e. how many rules are produced in total. Note that there exist duplicated rules. Table 6.2 shows the actual size of the learned rule sets with duplicates removed. Each row of Table 6.3 corresponds to the performance of the weighted rule set learned after the number of iterations specified by that row. Since ITR and ITR-H learned 5 rules for each ranking problem generated in each iteration, we compared the results after every 5 rules being induced.

For example, the first row for Blocksworld corresponds to the weighted rule set learned after 5 rules are induced. However, after removing duplications, the actual size of the weighted rule set is 3 for RB and 4 for RB-H. The next row indicates that the size of the weighted rule set is 7 for RB after 10 rules are induced.

Performance Across Different Learning Approaches. Table 6.1 compares the performance of different planners. First, note that, Yoon08 is unable to solve any problems in the three IPC4 domains, which suggests that learning control knowledge for greedy search is non-trivial in these problems. In particular, one form of control knowledge considered in [Yoon *et al.*, 2008a] were reactive rule-based policies. Rather we see that our weighted rule sets are often able to solve a non-trivial number of problems in these domains. This gives some evidence that using weighted rule sets is a more robust approach for learning and using action-selection rules.

For Blocksworld, ITR is the best planner and solves all problems with best median plan length. RB, RB-H and ITR-H also solve all problems indicating that

Table 6.2: The number of unique rules learned by different approaches. For each domain, we show the number of unique rules that are learned after the corresponding number of learning iterations.

	Learning iterations	Number of unique rules			
		RB	RB-H	ITR	ITR-H
Blocksworld	5	3	4	5	5
	10	7	8	8	10
Depots	5	4	5	5	4
	10	7	9	8	8
	15	9	13	10	9
	20	11	17	12	12
	25	13	20	14	14
	30	16	24	16	15
DriverLog	5	4	5	4	5
	10	6	5	5	8
	15	7	6	6	10
	20	8	8	6	10
	25	9	9	6	10
	30	10	11	6	10
FreeCell	5	2	4	2	5
	10	4	7	2	5
	15	7	10	2	6
	20	11	15	2	6
	25	11	19	2	6
Pipesworld	5	3	4	3	3
	10	7	7	7	7
	15	9	9	9	9
	20	9	12	10	13
	25	11	16	11	15
	30	12	19	11	18
Pipesworld-with-tankage	5	5	5	4	4
	10	6	8	5	9
	15	9	9	5	10
	20	12	12	7	10
	25	16	17	7	10
Philosopher	5	3	3	3	3
	10	3	3	4	4
	15	4	5	4	4
	20	5	5	5	4)

Table 6.3: Experimental results for the learned weighted rule sets. The performance of each learned rule set is given by the number of solved problems and the median plan length of the solved problems. A dash in the table indicates that the median plan length is not available while none of the problems can be solved.

	Learning iterations	Problems solved (Median plan length)			
		RB	RB-H	ITR	ITR-H
Blocksworld	5	30(133)	30(151)	**30(125)**	30(160)
	10	30(126)	30(166)	**30(89)**	30(118)
Depots	5	0(-)	2(8631)	0(-)	**3(115)**
	10	3(9194)	4(954)	0(-)	**20(796)**
	15	5(5372)	2(113)	0(-)	**16(313)**
	20	2(5193)	7(263)	0(-)	**23(433)**
	25	3(3188)	5(678)	0(-)	**22(349)**
	30	15(661)	11(129)	0(-)	**19(314)**
DriverLog	5	0(-)	1(1893)	0(-)	**3(8932)**
	10	0(-)	**3(2852)**	0(-)	1(2818)
	15	0(-)	0(-)	0(-)	**3(544)**
	20	0(-)	1(4309)	0(-)	**4(544)**
	25	0(-)	3(4082)	0(-)	**4(544)**
	30	0(-)	1(632)	0(-)	**3(544)**
FreeCell	5	2(213)	6(104)	2(213)	**9(94)**
	10	2(186)	5(112)	2(213)	**7(95)**
	15	3(103)	6(143)	2(213)	**9(94)**
	20	5(155)	7(96)	2(213)	**9(94)**
	25	3(334)	5(90)	2(213)	**9(92)**
Pipesworld	5	4(382)	**17(1063)**	7(1572)	16(279)
	10	2(7845)	8(1821)	2(335)	**11(307)**
	15	5(2929)	12(1599)	2(335)	**17(595)**
	20	3(1369)	11(1423)	5(511)	**17(579)**
	25	4(998)	11(2561)	6(883)	**17(595)**
	30	7(1360)	12(1423)	6(990)	**15(366)**
Pipesworld-with-tankage	5	0(-)	3(296)	3(2006)	**4(126)**
	10	0(-)	3(100)	1(5372)	**4(148)**
	15	0(-)	**4(98)**	1(4735)	4(134)
	20	1(1383)	**6(152)**	1(4735)	5(350)
	25	1(1383)	4(449)	3(2940)	**5(206)**
Philosopher	5	0(-)	**33(363)**	0(-)	**33(363)**
	10	0(-)	**33(363)**	33(875)	0(-)
	15	0(-)	**33(363)**	33(875)	0(-)
	20	33(875)	**33(363)**	33(875)	0(-)

the rule-based control knowledge in this domain is more useful than the relaxed plan length heuristic and the previously learned heuristic. Similar results are shown for Philosopher. For other domains except for Pipesworld-with-tankage, the best planner is ITR-H, solving more problems with fairly good plan length. In Pipesworld-with-tankage, RPL outperforms our planners. But RB-H solves the same number of problems, with the plan quality being a little worse.

The results show that the learned weighted rule sets significantly outperform the relaxed plan length heuristic and the heuristic in LaSO-BR$_1$ that only captures information about states. Overall, ITR-H is the best approach and to the best of our knowledge, these are the best reported results of any method for learning control knowledge for greedy search.

Performance Across Learning Iterations. Table 6.2 and 6.3 gives more detailed results of our learning approaches. As the learning goes on, the number of rules produced will be non-decreasing. However, since there exist duplicate rules, the size of the weighted rule set may not change. For example, ITR learned only 2 unique rules for FreeCell, regardless of how many times the rule learner is called. This either indicates a failure of our rule learner to adequately explore the space of possible rules, or indicates a limitation of our language for representing rules in this domain. These issues will be investigated in future work.

Note that in general, with some exceptions, the planning performance judged in terms of solved problems and median plan length improves as the number of unique rules increases. For example, RB solves 3 problems with 13 rules but 15 problems with 16 rules for Depots. As an exception, however, consider Philosopher, where

ITR-H solves all problems with the first 3 rules learned. When one new rule is added, it can not solve any of those problems. It is very likely that our weighted rule set converges to bad local minima, either because of the weight learning algorithm or the iteratively boosting algorithm, or both.

Iterative Learning vs. Non-iterative learning. ITR and ITR-H can be viewed as an iterative version of RB and RB-H, respectively. In general, with some exceptions, the iteration versions work better than the non-iterative versions, particularly for ITR-H. For Blocksworld, all of them have similar performance, while ITR improves the plan length over RB. For Depots, the iterative version ITR fails to solve any problem but contrastingly, ITR-H works much better than non-iterative version RB-H. For other domains, the iterative versions often achieve a better performance.

Effect of Relaxed Plan Length Heuristic. The only difference between ITR-H and ITR, as well as the difference between RB-H and RB, is that we used the relax plan length heuristic as prior knowledge for the former methods. Table 6.3 shows that the relaxed plan length heuristic did help to significantly improve performance in some domains. For DriverLog, FreeCell, Pipesworld and Pipesworld-with-tankage, RB-H and ITR-H solve more problems than RB and ITR, with the same number of rules induced. In Depots, RB-H achieves similar performance as RL but ITR-H solves many more problems than ITR. Blocksworld is a domain where ITR and RB work slightly better than ITR-H and RB-H, with better solution length. In Philosopher, ITR-H and RB-H have better solution path. Overall, there is clear value in using the relaxed-plan length heuristic as prior knowledge.

Chapter 7 – Learning Features and Weights for Randomized Greedy Search

It has been shown in Chapter 6 that the learned weighted rule sets greatly improve the performance of greedy search. One limitation of this approach, however, is that the search process is deterministic. When the search procedure gets trapped in a wrong path, it can not detect the error and will follow the path until reaching the time limit. To solve this problem, we consider randomized greedy search with restarts in this chapter.

7.1 Introduction

For incomplete search algorithms, it is often helpful to add randomization and allow quick restarts for solving problems. For example, WalkSat is a solver for satisfiability problems, which repeats the process of randomly picking an initial assignment and conducting a randomized local search [Selman *et al.*, 1995]. By adding randomization to complete search algorithms, the search engine also achieves better performance [Gomes, 1998]. We were motivated by the success of this prior work and studied the problem of learning for randomized greedy search in this chapter.

Given a ranking function F, instead of selecting the highest ranked node according to F as described in Chapter 6, we calculate a probability distribution and select actions based on the distribution. The state transitions with higher ranks will have higher probabilities of being selected. The ranking function F can thus be viewed as a randomized policy. When the randomized policy fails to solve a planning problem within a certain amount of time, we will restart it and another run of the randomized policy will probably generate a different path.

The learning objective here is to learn randomized policies that can guide randomized greedy search with restarts to efficiently solve problems. We investigated this learning problem in the framework of reinforcement learning and learned features and weights using recent policy-gradient algorithms [Baxter and Bartlett, 2000; Baxter *et al.*, 2001; Kersting and Driessens, 2008]. Similar to prior learning approaches presented in the dissertation, we tightly integrated learning with search, aiming to correct observed search errors.

The remainder of the chapter is organized as follows. First, we formulate our learning problem for learning randomized policies. Then we present the adaptive weight learning and feature learning algorithms for our planning application. We finally present the experimental results and conclude.

7.2 Problem Setup

In this section, we first give the definition of the randomized policy and then formulate our learning problem in the reinforcement learning framework.

7.2.1 Randomized Policy Representation

We extend our linear ranking function, which is represented as a set of rule-based features, to a randomized policy. Given a ranking function $F(\omega, g, a) = \sum_i w_i f_i(\omega, g, a)$, we calculate a probability distribution as below.

$$\pi((\omega, g), a) = \frac{\exp(F(\omega, g, a))}{\sum_{a'} \exp(F(\omega, g, a'))}$$

Here $\pi(v, a) = Pr(a|v)$ is the probability of selecting action a from search node $v = (\omega, g)$. The ranking function F could thus be viewed as defining a randomized policy that will direct the randomized greedy search. At each step of the search process, the policy selects the action according to the probability distribution π. Therefore, different runs of the randomized policy can result in different search paths. We set a depth limit for the randomized greedy search. Whenever this limit is reached and no solution is found yet, we will start a new run of the randomized greedy search.

7.2.2 Reinforcement Learning Formulation

The field of reinforcement learning (RL) studies algorithms that interact with the environment and observe the rewards to learn control knowledge. A basic model for RL consists of a set of states S, a set of actions A, a transition function T and a reward function R. Following the discussion in Section 5.1.2, each state $s \in S$ is a state-goal pair (ω, g), where ω is a planning state and g represents the goal in the current planning problem. The set of actions $A = \mathcal{A} \cup \{\epsilon\}$ where $\mathcal{A}$ contains all possible actions in the given planning domain and ϵ is a special action used in learning. For any node that is not on the target solution paths, applying action ϵ will lead to one of its sibling nodes that are on the target solution paths.

The transition function T gives the probability of transitioning to a state by taking an action in the current state. We define the transition function T as below.

- for any target node $v = (s, g)$,

  $T(v, a, v') = 1$, if a is an applicable action in s and $v' = (s', g)$ where s' is the state generated by applying a in s;

  $T(v, a, v') = 0$, otherwise;

- for any non-target node $v = (s, g)$,

  $T(v, a, v') = 1/k$, if $a = \epsilon$ and v' is one of the k sibling nodes of v that are on the target solution paths;

  $T(v, a, v') = 0$, otherwise.

In the learning process, ϵ is the only action that can be taken from the non-target nodes. This forces our learning algorithm to follow the target solution paths in the training set. Accordingly, we define the reward function R.

- for any target node $v = (s, g)$,

  $R(v, a) = k_+ > 0$, if applying action a in s results in a target node;

  $R(v, a) = k_- < 0$, otherwise;

- for any non-target node $v = (s, g)$

  $R(v, a) = 0$.

The reward function R assigns an immediate reward for every transition, which is positive if the transition starts from a target node and also generates a target node, and negative if the transition starts from a target node and generates a non-target node. With this reward function, the learned randomized policy will tend to select the nodes on the target solution paths to maximize its expected average reward.

7.3 Gradient-based Approach for Weight Learning

Given the above RL formulation, we first assume that the features in the ranking function have already been provided. Our learning objective is to learn a weight vector that can maximize the expected average reward. One algorithm for doing this is GPOMDP [Baxter and Bartlett, 2000], which estimates the gradient of the average reward and then uses the gradient estimation to update the weight vector. We consider its online version, which is called OLPOMDP [Baxter *et al.*, 2001], and apply this algorithm to our planning application.

Figure 7.1 shows the OLPOMDP algorithm for our planning application. Each training instance here is a planning problem $x_i = (s_{i0}, A, g_i)$. The learning algorithm resembles LaSO-BR$_1$ in the sense that it iterates through all training instances and conducts greedy search from the initial node (s_{i0}, g_i). Based on the transition function we defined, whenever the selected action leads to a non-target node, this node will be replaced by one of its siblings that are target nodes. We force the search process to follow an example solution path.

```
**OLPOMDP** $(\{x_i\}, \delta, H)$
// $x_i = (s_{i0}, A, g_i)$ : planning problem
// $\delta$ : partial-order plan indicator function
// $H$ : the given ranking function
repeat for a number of iterations or until no improvement
    for each $x_i = (s_{i0}, A, g_i)$
        $s \leftarrow s_{i0}$
        $\Delta = 0$
        while $s$ is not a goal node
            draw an action $a$ according to $\pi((s, g), a)$
            observe the reward $R$
            $\Delta = \beta\Delta + \nabla_w log(Pr(a|(s, g), w))$
            $w \leftarrow w + \alpha R\Delta$
            $s \leftarrow$ the state generated by applying action $a$ on state $s$
            if $\delta(x_i, s) = false$      // $s$ is not on the example solution paths
                $s \leftarrow$ randomly selecting a sibling $s'$ of $s$ that satisfies $\delta(x_i, s') = true$
return $F$
```

Figure 7.1: The OLPOMDP algorithm.

The difference between this gradient-based RL algorithm and LaSO-BR$_1$ is in weight updating. As a supervised learning algorithm, LaSO-BR$_1$ moves the weights toward the direction of the target nodes and away from the non-target nodes. However, the gradient-based RL algorithm updates the weights by estimating the gradient of the average reward. It must compute the gradient $\nabla_w \log(Pr(a|(s, g), w))$, which for our policy representation is given by

$$
\begin{aligned}
& \nabla_{w_i} \log(Pr(a|(s,g),w)) \\
= \ & \nabla_w F(s,a) - \nabla_w \log(\sum_{a'} \exp(F(s,g,a'))) \\
= \ & -h_i(s,g,a) - \frac{\sum_{a'} \exp(F(s,g,a'))(-h_i(s,g,a'))}{\sum_{a'} \exp(F(s,g,a'))} \\
= \ & -h_i(s,g,a) + \sum_{a'} \pi_w(s,g,a') h_i(s,g,a')
\end{aligned}
$$

Computing all gradients along a search path, we can then estimate the gradient of the average reward by the discounted sum of these gradient directions, as shown in Figure 7.1. This is guaranteed to find a local optimum [Baxter and Bartlett, 2000].

7.4 Learning Weighted Rule Sets by Policy Gradients

In the previous section we have shown a gradient-based algorithm for tuning the weights of given features. These features, as we considered, are the rule-based features introduced in Section 6.2. In addition to weight learning, we are also interested in inducing these features directly to maximize the expected average reward. Here we apply a non-parametric policy gradient approach, called NPPG [Kersting and Driessens, 2008], to find the locally optimal policies.

The key idea of NPPG is to estimate the policy gradient by a regression function learned from the training set $\{(v,a), f_m(v,a)\}$. Here v is a state-goal pair (s,g)

where s is a state on the search path that is generated by executing policy π, a is an action applicable in state s. $f_m(v,a)$ is the point-wise functional gradient at (v,a), which can be computed as

$$f_m(v,a) = \frac{\partial \pi(v,a)}{\partial F} Q(v,a)$$

where $Q(v,a)$ is the unknown discounted state-action value. One way to estimate $Q(v,a)$ is by using the Monte Carlo methods. In this work, we focus on the immediate reward and use $\beta = 0$. So $Q(v,a) = R(v,a)$. According to Proposition 3.1 of [Kersting and Driessens, 2008],

$$\frac{\partial \pi(v,a)}{\partial F(v,a)} = \pi(v,a)(1-\pi(v,a)).$$

Provided with the training set $\{(v,a), f_m(v,a)\}$, the NPPG algorithm then aims to learn a function that will minimize the squared error and updates the policy in the direction of the learned f_m. Figure 7.2 shows the revised NPPG algorithm for our planning application.

As shown in Figure 7.2, the NPPG algorithm starts with an initial function and iteratively adds functional gradients to correct the errors. In each iteration, it constructs a training set based on the point-wise functional gradients and then trains a function so that it minimizes the loss over the training examples. We also apply OLPOMDP to optimize the feature weights learned in the NPPG algorithm. Again, we force the policy to select an action that will lead to a target node, i.e. a node s with $\delta(x,s) = \mathit{true}$ where x is the current planning problem. Whenever

```
NPPG ($\{x_i\}, \delta, H$)
// $x_i = (s_{i0}, A, g_i)$ : planning problem
// $\delta$ : partial-order plan indicator function
// $H$ : the given ranking function
$F = -H$
for $m = 1$ to $N$ do
    $\Sigma \leftarrow \emptyset$
    for each $x_i = (s_{i0}, A, g_i)$
        $s \leftarrow s_{i0}$
        while $s$ is not a goal node
            compute the probabilities $\pi_{m-1}((s,g),a)$ for all $a$
                $\pi_{m-1}((s,g),a) = \frac{exp(F(s,g,a))}{\sum_{a'} exp(F(s,g,a'))}$
            observe the immediate reward $R(s,g,a)$ for all $a$
            compute the point-wise functional gradients
            $f(s,g,a) = \pi_{m-1}((s,g),a)(1 - \pi_{m-1}((s,g),a))R(s,g,a)$
            $\Sigma \leftarrow \Sigma + \{((s,g), a, f(s,g,a))\}$
            draw an action $a$ according to $\pi_{m-1}$
            $s \leftarrow$ the state generated by applying $a$ on $s$
            if $\delta(x_i, s) = false$
                $s \leftarrow$ a sibling $s'$ of $s$ where $\delta(x_i, s') = true$
    //Induce a feature from $\Sigma$ that minimizes the loss $\Phi$
    $f_m \leftarrow$ Rule-Learner($\Sigma$)
    $w_m \leftarrow \text{argmin}_w \sum_{s,g,a} \Phi(f, wf_m)$
    $F \leftarrow F + w_m f_m$
    $F \leftarrow$Weightlearning($F$) //OLPOMDP($\{x_i\}, \delta, -F$)
return $F$
```

Figure 7.2: The revised NPPG algorithm.

the policy selects an action that results in a non-target node, this node will be replaced by one of its siblings that satisfy $\delta(x, s) = true$.

After constructing the training set Σ, we need to learn feature f_m and its corresponding weight w_m that minimize the loss Φ on the training examples. In this work, we consider each feature as an action selection rule and use the rule learner introduced in Section 6.3.5 to learn the feature. The procedure **Rule-Learner** performs a beam search over possible rules, where the starting rule has no literals in its body and each search step adds one literal to the body. The search terminates when the beam search is unable to further decrease the squared error.

7.5 Experimental Results

As in Section 6.4, we present experiments in seven STRIPS domains: Blocksworld, Depots, DriverLog, FreeCell, Pipesworld, Pipesworld-with-tankage and Philosopher. We set a time cut-off of 30 CPU minutes and a maximum search depth of 1000. Once the depth 1000 is reached and the randomized greedy search has not found a solution, we will terminate it and restart a new randomized greedy search from the initial node. If a solution is not found within the time cut-off of 30 CPU minutes, we considered the problem to be unsolved.

In our experiments, we use the reward function $R(v, a) = 1$ for any transition (v, a) that is on the partially ordered plan and $R(v, a) = -5$ otherwise. Note that this reward function considers all nodes on the partially ordered plan to be of equal value, regardless of their distance to the goal. Therefore, these nodes are viewed as independent instances without sequential dependence. For this reason we set $\beta = 0$ and $Q(s, a) = R(s, a)$ which tell the algorithm to consider only the immediate reward.

We use the NPPG algorithm to automatically induce new rules. The learning algorithm terminates when no new rule can be induced or the maximum number of rules has been reached. We discovered that our rule learner often outputs duplicated rules. Here we set the maximum number of learned rules to be 5, since for most domains, the rule learner already fails to induce any new rule among the first 5 rules. We have learned 4 rules in Blocksworld, 2 rules in Pipesworld, 5 rules in Depots, Driverlog and Pipesworld-with-tankage. For FreeCell and Philosopher, we learn only 1 unique rule.

Description of Tables. Table 7.1 compares the performance of the learned randomized policies with the ranking function learned in Chapter 6. For each of the randomized policy or ranking function, it is a weighted rule set and could be evaluated in two different ways. First, we use it as a ranking function to guide deterministic greedy search, with the highest ranked node being selected at each search step, which is the evaluation approach in Chapter 6. Second, we use it as a randomized policy to guide randomized greedy search with restarts, as we described in this chapter. The algorithms in Table 7.1 are:

- ITR-H : greedy search with the weighted rule set that is learned via our iterative learning approach starting with the relaxed-plan length heuristic as prior knowledge. The details of this algorithm are given in Chapter 6.

- ITR-H^R: greedy randomized search with restarts guided by the same ranking function in ITR-H. Here the ranking function is used as a randomized policy.
- GW: greedy search with the weighted rule set that is learned via OLPOMDP. This learning approach conducts only weight learning and assumes the features are already provided. We use the set of features learned in ITR-H. Therefore, the difference between GW and ITR-H is that the weights in GW are learned by the revised OLPOMDP algorithm.
- GW^R: greedy randomized search with restarts guided by the same weighted rule set in GW, which is now used as a randomized policy.
- GL: greedy search with the weighted rule set that is learned via the revised NPPG algorithm. We also use the relaxed-plan length heuristic as prior knowledge.
- GL^R: greedy randomized search with restarts guided by same weighed rule set in GL, which is now used as a randomized policy.

Each row of Table 7.1 corresponds to a target planning domain and each column corresponds to an algorithm. The planning performance is first evaluated on the number of solved problems. When two algorithms solve the same number of problems, we use the median plan length of the solved problems to break the tie. The one that has shorter plans is considered better.

Performance Evaluation. From Table 7.1 we can see that in a number of domains, adding randomization directly to the previously learned ranking function

Table 7.1: Experimental results for the learned randomized policies. 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 median plan length is not available while none of the problems can be solved.

Problems solved (Median plan length)	ITR-H	ITR-H^R	GW	GW^R	GL	GL^R
Blocksworld	**30(118)**	0(-)	30(129)	30(137)	11(536)	28(572)
Depots	**23(433)**	0(-)	11(847)	21(331)	2(61)	18(315)
DriverLog	4(544)	0(-)	0(-)	**6(724)**	4(6408)	4(877)
FreeCell	9(92)	17(242)	9(92)	21(141)	6(107)	**25(122)**
Pipesworld	17(579)	11(596)	17(2315)	20(179)	12(162)	**26(303)**
Pipesworld-with-tankage	5(206)	1(359)	6(158)	**12(211)**	4(104)	11(262)
Philosopher	**33(363)**	4(234)	0(-)	26(402)	0(-)	20(435)

hurts the performance. For example, with the same weighted rule set, ITR-H solves all problems in Blocksworld but ITR-H^R can not solve even a single problem. For the Depot domain, ITR-H solves 23 problems and ITR-H^R again solves none of the problems. The only exception here is the FreeCell domain, in which randomized greedy search with the same weighted rule set solves about twice as many problems as greedy search. These results are in contrast to the following results of the learned randomized policies, showing the effectiveness of learning randomized policies for guiding randomized greedy search.

The columns labeled as GW and GW^R show the performance of the randomized policy learned by weight learning. With the same set of features as in ITR-H, the weights of GW and GW^R are learned by the gradient approaches in the framework of reinforcement learning. For Blocksworld , GW^R now solves all 30 problems with comparable plan length, compared to ITR-H. For Driverlog, FreeCell, Pipesworld and Pipesworld-with-tankage, GW^R outperforms ITR-H with more planning problems being solved. For the other two domains, GW^R solves fewer problems, but still is comparable with ITR-H. However, GW always solves fewer problems than GW^R and is usually worse than ITR-H. This indicates that our weight learning algorithm has successfully learned a good weight vector for better guiding randomized greedy search.

The last two columns in Table 7.1 show the performance of GL and GL^R which learn both features and weights. For FreeCell and Pipesworld, GL^R has the best performance in Table 7.1. It solves 25 out of 35 problems in FreeCell and 26 out of 35 problems in Pipesworld. For all other domains, however, it solves fewer problems than GW^R. This suggests that our feature learning mechanism still has significant room to improve.

Chapter 8 – Summary and Future Work.

8.1 Summary of Contributions

This dissertation presented a detailed study of the problem of learning ranking functions for efficient search in automated planning. We considered ranking functions represented as a linear combination of features and aimed to learn features and weights that can guide beam search, greedy search, or randomized greedy search to perform nearly as well as unconstrained search.

First, we investigated the problem of learning weights for linear ranking functions to guide beam search. On the theoretical side, we studied the computational complexity of this learning problem, highlighting the main dimensions of complexity by identifying core tractable and intractable subclasses. We also studied the convergence of recent online learning algorithms for this problem. The results clarified convergence issues, correcting and extending 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.

Next, we studied the problem of automatically learning both features and weights to guide greedy search. We introduced weighted sets of action-selection rules as a new form of control knowledge, which allow multiple rules to vote, helping to improve robustness to noisy rules. We also presented a new iterative learning algorithm for learning weighted rule sets based on RankBoost, an efficient boosting algorithm for ranking. Each iteration considers the actual performance of the current rule set and directs learning based on the observed search errors. This is in contrast to most prior approaches, which learn control knowledge independently of the search process. Our empirical results have shown significant promise for this approach in a number of domains.

Finally, we considered randomizing the search process and learning randomized policies for guiding randomized greedy search with restarts. We investigated the learning problem in the framework of reinforcement learning and applied recent policy-gradient algorithms for learning features and weights. Our experiments show that in a number of domains, randomized greedy search has better performance than deterministic greedy search, especially with the guidance of our learned randomized policies.

8.2 Future Work

This dissertation has shown significant promise for learning ranking functions for efficient search. Beyond the research work discussed here, there are several interesting problems that require further investigation.

Theoretical Analysis. It is important to study the theoretical properties of our learning algorithms, especially for feature learning. In Chapter 4, we have presented a detailed study of the theoretical properties of the weight learning algorithms. But for feature learning in Chapter 6, we only show a worst case bound. We are interested in investigating new assumptions that can guarantee the convergence in a small number of iterations.

Moreover we are interested in understanding the fundamental limitations on learning for greedy search. In particular, can one characterize when it is possible to practically compile search away via learning? One way to begin addressing this question is to understand when it is and is not possible to succinctly represent greedy control knowledge for a search problem.

Improving the Weak Learner. The experiments in Chapter 6 and 7 have shown that our rule learner often generates duplicated rules. For example, the revised NPPG algorithm fails to induce any new rule after a single rule is learned in the Philosopher domain. We suspect that our results can be further improved by using more powerful weak learning algorithms, e.g. relational decision trees as in [de la Rosa *et al.*, 2008].

Combining Different Types of Features. Finally, we plan to learn other types of features in planning domains. One option is to consider reactive policies, which consist of a list of action-selection rules. Similar to action-selection rules, the reactive policies are evaluated on state transitions and have binary values. The main difference is that reactive policies can represent more complex control knowledge. It is possible to extend our work in Chapter 6 to learning weighted policies by simply replacing the weak rule learner by a policy learner.

In the application of weight learning, we have considered features as taxonomic class expressions. These features, unlike the binary rule-based or policy-based features, have integer values which increases the difficulty of learning these features. We will also attempt to automatically learn this type of features, so that our ranking function can represent more powerful control knowledge.

Bibliography

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

[Ambite *et al.*, 2000] Jose Luis Ambite, Craig A. Knoblock, and Steven Minton. Learning plan rewriting rules. In *Artificial Intelligence Planning Systems*, pages 3–12, 2000.

[Baxter and Bartlett, 2000] Jonathan Baxter and Peter L. Bartlett. Reinforcement learning in POMDP's via direct gradient ascent. In *In Proc. 17th International Conf. on Machine Learning*, pages 41–48. Morgan Kaufmann, 2000.

[Baxter *et al.*, 2001] Jonathan Baxter, Peter L. Bartlett, and Lex Weaver. Experiments with infinite-horizon, policy-gradient estimation. *Journal of Artificial Intelligence Research*, 15:2001, 2001.

[Bonet and Geffner, 1999] Blai Bonet and Hector Geffner. Planning as heuristic search: New results. In *European Conference on Planning*, pages 360–372, 1999.

[Daumé III and Marcu, 2005] H. Daumé III and Daniel Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

[Daumé III *et al.*, 2009] Hal Daumé III, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine Learning*, 75:297–325, 2009.

[de la Rosa *et al.*, 2008] Tomás de la Rosa, Sergio Jiménez, and Daniel Borrajo. Learning relational decision trees for guiding heuristic planning. In *ICAPS*, pages 60–67, 2008.

[Estlin and Mooney, 1996] Tara Estlin and Raymond Mooney. Integrating EBL and ILP to acquire control rules for planning. In *International Workshop on Multi-Strategy Learning*, 1996.

[Fern *et al.*, 2006] 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.

[Fikes *et al.*, 1972] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. 3(1–3):251–288, 1972.

[Freund *et al.*, 2003] Yoav Freund, Rai Iyer, Robert E. Schapire, and Yoram Singer. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, 4:933–969, 2003.

[Garey and Johnson, 1979] 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.

[Gomes, 1998] Carla P. Gomes. Boosting combinatorial search through randomization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 431–437. AAAI Press, 1998.

[Hoffgen *et al.*, 1995] 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.

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

[Huang *et al.*, 2000] 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.

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

[Kersting and Driessens, 2008] Kristian Kersting and Kurt Driessens. Non-parametric policy gradients: A unified treatment of propositional and relational domains. In *Proceedings of the 25th international conference on Machine learning*, 2008.

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

[Khardon, 1999] Roni Khardon. Learning action strategies for planning domains. *Artificial Intelligence*, 113:125–148, 1999.

[Kolobov *et al.*, 2009] Andrey Kolobov, Mausam Mausam, and Daniel S. Weld. ReTrASE: integrating paradigms for approximate probabilistic planning. In *IJCAI*, pages 1746–1753. Morgan Kaufmann Publishers Inc., 2009.

[la Rosa *et al.*, 2007] Tomas De la Rosa, Daniel Borrajo, and Angel Garca Olaya. Using cases utility for heuristic planning improvement. In *International Conference on Case Based Reasoning*, 2007.

[Martin and Geffner, 2000] 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.

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

[Minton *et al.*, 1989] 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.

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

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

[Nguyen *et al.*, 2002] XuanLong Nguyen, Subbarao Kambhampati, and Romeo Sanchez 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.

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

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

[Selman *et al.*, 1995] Bart Selman, Henry Kautz, and Bram Cohen. Local search strategies for satisfiability testing. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 521–532, 1995.

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

[Veloso *et al.*, 1991] 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.

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

[Xu *et al.*, 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.

[Xu *et al.*, 2009a] Yuehua Xu, Alan Fern, and Sungwook Yoon. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research*, 10:1367–1406, 2009.

[Xu *et al.*, 2009b] Yuehua Xu, Alan Fern, and Sungwook Yoon. Learning weighted rule sets for forward search planning. *Workshop on Planning and Learning in ICAPS-09*, 2009.

[Xu *et al.*, 2010] Yuehua Xu, Alan Fern, and Sungwook Yoon. Iterative learning of weighted rule sets for greedy search. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling*, 2010.

[Yoon *et al.*, 2002] 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.

[Yoon *et al.*, 2006] Sungwook Yoon, Alan Fern, and Robert Givan. Learning heuristic functions from relaxed plans. In *ICAPS*, 2006.

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

[Yoon *et al.*, 2008b] Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *AAAI*, 2008.

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

APPENDICES

Relation to Structured Classification

This Appendix assumes that the reader is familiar with the material in Section 4.2. The learning framework introduced in Section 4.1 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 rank 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 4.1, 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 in [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 rank 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 4.1 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 7 (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, $B^{(1)}_{i,d} \in P_{i,d}$, and $B^{(1)}_{i,j} \notin \mathcal{T}$ for $j < d$?*

Definition 8 (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 4.2 proves that the consistency problems are also NP-complete we immediately get equivalence.

Theorem 11 *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 1. 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 7. From Lemma 1 in Section 4.2 we can use the algorithm *TestTrajectories* in Figure 4.3 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 Table 4.1 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 12 *Best-first structured classification is NP-complete.*