Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS 2010)

Iterative Learning of Weighted Rule Sets for Greedy Search

Yuehua Xu
School of EECS
Oregon State University
xuyu@eecs.oregonstate.edu

Alan Fern
School of EECS
Oregon State University
afern@eecs.oregonstate.edu

Sungwook Yoon
Palo Alto Research Center
sungwook.yoon@parc.com

Abstract

Greedy search is commonly used in an attempt to generate solutions quickly at the expense of completeness and optimality. In this work, we consider learning sets of weighted action-selection rules for guiding greedy search with application to automated planning. We make two primary contributions over prior work on learning for greedy search. First, we introduce weighted sets of action-selection rules as a new form of control knowledge for greedy search. Prior work has shown the utility of action-selection rules for greedy search, but has treated the rules as hard constraints, resulting in brittleness. Our weighted rule sets allow multiple rules to vote, helping to improve robustness to noisy rules. Second, we give 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.

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. 2008; Kolobov, Mausam, and Weld 2009). Greedy search is one approach to finding solutions quickly by pruning away most nodes in a search space. However, when the guiding heuristic is not accurate enough, it often leads to failure. Here we focus on the problem of learning control knowledge 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, Fern, and Givan 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 paper, 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, allowing 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, which 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.

Other recent work (Yoon, Fern, and Givan 2008; de la Rosa, Jiménez, and Borrajo 2008) has made progress on learning various forms of control knowledge including heuristics and policies for guiding search in planning. That work does not focus on guiding greedy search[1] and as will be shown in our experiments, the knowledge learned in (Yoon, Fern, and Givan 2008) does not work well when applied greedily. Furthermore, the knowledge in those studies is learned completely independently of the search process. In particular, there is no mechanism for directly trying to improve the knowledge to correct for observed search errors. In contrast, the learning approach in this paper is directly driven by observed search errors, focusing on parts of the search space that are observed to be most difficult.

Recent studies have considered a tighter integration of search and learning in the context of structured prediction (Daume III and Marcu 2005) and automated planning (Xu, Fern, and Yoon 2009). However, these approaches only learn the weights of a human provided set of features and did not provide a feature learning mechanism. Rather in our work we consider learning weights and features, where here action-selection rules serve as the feature representation. 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, Langford, and Marcu 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 paper is organized as follows. First, we give our problem setup for learning control knowledge to guide greedy search. Second, we introduce the form of our rule-based ranking function. Next, we present a new iterative learning algorithm for learning the ranking function, followed by a description of our rule learner. We finally present experimental results and conclude.

Problem Setup

We focus our study on search problems arising from deterministic planning problems. A planning problem is a tuple (s_0, A, g), where s_0 is the initial state, A is a set of actions, and g is a set of state facts representing the goal. A solution plan for a planning problem is a sequence of actions $(a_1, \ldots, a_l)$, where the sequential application of the sequence starting in state s_0 leads to a goal state s^* where $g \subseteq s^*$. A planning problem can be viewed as searching a large directed graph where the vertices represent states and the edges represent applicable actions.

Greedy Search. We focus on planning scenarios where problems must be solved very quickly in order for their solutions to be useful. For this purpose we consider greedy search as our mechanism for generating solutions quickly. In particular, we assume access to a ranking function that assigns a numeric score to any possible state transition. Ties can be broken according to some canonical ordering (e.g. lexicographic). At each step of greedy search, the current search node is expanded and the child of the transition with the highest rank is selected to be the current node. This process continues until a goal node is discovered or time runs out.

Learning to Plan. Given a target planning domain, our goal in this paper is to learn a ranking function on state transitions that can quickly solve problems in the domain using greedy search. A learning-to-plan problem provides a training set of pairs $\{(x_i, y_i)\}$, where each $x_i = (s_i, A, g_i)$ is a planning problem from the target planning domain and each $y_i = (a_{i1}, a_{i2}, \ldots, a_{il})$ is a sequence of actions that achieves the goal g_i starting from initial state s_i. Here we have assumed, without loss of generality, that all solutions have length l to simplify notation. Given such a training set the learning goal is to learn a ranking function so that when used to rank state transitions, greedy search will solve the training problems, typically finding solutions that are the same or variants of the provided solutions in the training set. There is an implicit assumption that the set of training problems are representative of the problem distribution to be encountered in the future, so that learning will be biased toward the most relevant problem space. In addition, the solutions in the training set should reflect good solutions, since learning will be largely driven to produce variants of those solutions.

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, Fern, and Givan 2008) that used taxonomic syntax to define action-selection rules for reactive policies, each of our action-selection rules has the form:

$$u(z_1, \ldots, z_k) : L_1, L_2, \ldots, L_m \tag{1}$$

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.

[1]One approach in (de la Rosa, Jiménez, and Borrajo 2008) considers guiding enforced hill-climbing, which is more greedy than most search approaches, but still relies on unbounded (non-greedy) expansion at each step to find an improved node.

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. We refer the reader to (Yoon, Fern, and Givan 2008) for details of taxonomic syntax, noting that the details are not essential to the main contribution of this paper. 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

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

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.

Learning Weighted Rule Sets

In this section, we describe the traditional rank learning problem from machine learning and then how to 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.

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 $F(v_1) \leq F(v_2)$, or $(v_2, v_1) \in S$ and $F(v_1) \geq F(v_2)$. 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 paper, the instances will correspond to possible state transitions from a planning domain and our features will be the rule-based features described above.

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 transition 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, Pérez, and Carbonell 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 the 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 a good/consistent transition not in the totally ordered solution), where it has not been trained 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.

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, Fern, and Givan 2008; Xu, Fern, and Yoon 2009) 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 1, the main idea is to

```
**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 1: The variant of RankBoost.

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.

Theorem 1 *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, which we omit for space reasons, 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 $\epsilon > 0$.

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, which 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 2 gives pseudo-code for our improved approach to learning ranking functions for planning. The top level procedure repeatedly constructs 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 **WeightLearning** using a perceptron-style algorithm. 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 weight-learning algorithm iteratively conducts greedy search and updates the weights in order to correct the search errors that have been made. For details about the weight learning algorithm, we refer to prior work (Xu, Fern, and Yoon 2009).

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 partial order plan.

```
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 ($S, F, k$)
    $F \leftarrow F'$
    $F \leftarrow$ WeightLearning($F$)
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 2: The iterative learning algorithm.

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.

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 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, Fern, and Givan 2008) 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 $|r|$.

Experimental Results

Domains. To facilitate comparison we present experiments in seven STRIPS domains from prior work (Yoon, Fern, and Givan 2008; Xu, Fern, and Yoon 2009) that also studied learning knowledge from provided solution trajectories. The domains included: Blocksworld (30-30), Depots (15-35), Driverlog (15-35), FreeCell (15-35), Pipesworld(15-35), Pipesworld-with-tankage(15-35) and Philosopher(15-33), which had 30 training and 30 test problems for Blocksworld and for all other domains 15 training problems and 35 or 33 test problems. For each training problem, we select the shortest plan out of those found by running FF and beam search with various beam widths to be the solution trajectory. For consistency with those prior studies we set a time cut-off of 30 CPU minutes, upon which a problem was considered a failure, however, for our greedy search approach, when solutions are found they are found very quickly.

Description of Tables. Figure 3 compares the performance of our different learning approaches as well as to the performance of greedy search using control knowledge learned by various algorithms from prior work (Yoon, Fern, and Givan 2008; Xu, Fern, and Yoon 2009). The algorithms are: **Yoon08:** three forms of control knowledge were learned in (Yoon, Fern, and Givan 2008) (policies, heuristics, and measures-of-progress). The table entries labeled Yoon08 give the best performance among the three types of control knowledge when used for greedy search as reported in that work. Results for Yoon08 are only given for our three IPC4 domains 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$_1$ (Xu, Fern, and Yoon 2009). This is the closest work to our approach, in which the ranking function is also represented as a linear combination of features and used to control greedy search. The features in that work were features of state rather than transitions and were provided to the algorithm rather than learned as in this work. **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-prior:** same as RB but with the relaxed plan length heuristic provided as prior knowledge. **ITR:** greedy search with the weighted rule set that is learned via our iterative learning algorithm. In this experiment, we do not accumulate data across iterations as described in Figure 2, for efficiency reasons, but rather pass only the most recent data to RB-prior. Also, we implement only an approximation of the δ function since an exact test is computationally hard (see full report for details). We initialize the algorithm with the relaxed plan length heuristic as input in the first iteration and learns $k = 5$ rules with RP-prior per iteration. Learning is terminated when no improvement is observed or 30 rules are learned in total.

For an additional reference point, we also include the performance of FF (Hoffmann and Nebel 2001) in Figure 3 which is not constrained to perform greedy search. Each column of Figure 3 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.

Figure 4 provides more details of the approaches we used. We add a new column "Learning iterations" indicating how many times the rule learner is called. Since the learner sometimes learns duplicated rules, we add a set of three columns that are labeled as "Number of unique rules", giving the actual size of the learned rule set that removes duplications. Now each row corresponds to the performance of the weighted rule set learned after the number of iterations specified by that row. Since ITR 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-prior. The next row indicates that

Problems solved (Median plan length)	FF	Yoon08	RPL	LaSO-BR$_1$	RB	RB-prior	ITR
Blocksworld	10 (77)	N/A	13 (3318)	27 (840)	30 (126)	30 (166)	**30 (118)**
Depots	14 (63)	N/A	1 (462)	4 (1526)	15 (661)	11 (129)	**23 (433)**
Driverlog	3 (119)	N/A	0 (-)	0 (-)	0 (-)	3 (2852)	**4 (544)**
FreeCell	29 (90)	N/A	5 (96)	7 (132)	5 (155)	7 (96)	**9 (92)**
Pipesworld	20 (50)	0 (-)	11 (114)	16 (1803)	7 (1360)	17 (1063)	**17 (579)**
Pipesworld-with-tankage	3 (63)	0 (-)	**6 (119)**	5 (55)	1 (1383)	6 (152)	5 (206)
Philosopher	0 (-)	0 (-)	0 (-)	6 (589)	33 (875)	**33 (363)**	**33 (363)**

Figure 3: Experimental results for different planners. 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 since none of the problems can be solved. N/A indicates that the result of the planner is not applicable here.

the size of the weighted rule set is 7 for RB after 10 rules are induced.

Performance Evaluation

From Figure 3 we can see that overall among the learning approaches, ITR performs the best in all domains, solving more problems with fairly good plan length. Also, ITR is as good and in many cases better than the relaxed plan length heuristic (RPL) and LaSO-BR$_1$. From Yoon08 it is shown that no form of control knowledge learned in (Yoon, Fern, and Givan 2008) was able solve any problem when used to guide greedy search. Interestingly one form of knowledge was (unweighted) rule-based policies using an identical rule language to our own. This provides some evidence that using weighted rule sets is a more robust way of using rules in the context of greedy search. To the best of our knowledge, these are the best reported results of any method for learning control knowledge for greedy search. Finally, we see that ITR is comparable or better than FF, which is not restricted to greedy search, in all domains except for Freecell, where ITR is significantly outperformed. This shows that the performance of ITR is not due to easy problem sets.

Performance Across Learning Iterations. In Figure 4, we first make the observation that our rule learner often learns duplicate rules from previous iterations. In fact, in some cases, the learner fails to find new rules, e.g ITR stops learning new rules for Driverlog after 15 iterations, 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 solves all problems with the first 3 rules learned. When one new rule is added, it can not solve any of those problems. It appears that the rule learner is unable to learn new rules based on the training data that are able to take it out of the bad local minima. Again we suspect that this is a more likely a problem with the rule learner rather than with the overall approach, though this needs to be further investigated.

Effect of Prior Knowledge. The only difference between RB-prior and RB is that we used the relaxed plan length heuristic as prior knowledge for the former method. In Figure 4, it is shown that the relaxed plan length heuristic did help to significantly improve performance in some domains. Overall, there is a clear improvement on the planning performance that is achieved by adapting RankBoost to take advantage of prior knowledge.

Iterative Learning vs. Non-iterative learning. While RB-prior outperforms the original RankBoost, our iterative learning algorithm has even better performance. ITR can also be viewed as an iterative version of RB-prior. In general, ITR works better than RB-prior, solving more planning problems and improving the plan length. The only exception is in the Pipesworld-with-tankage domain, where RB-prior solves one more problem than ITR.

Summary and Future Work

We developed a new approach for learning to control greedy search. First, we introduced a new form of control knowledge, weighted rule sets, as a more robust way of using action-selection rules than prior work. Second, we developed a learning algorithm for weighted rule sets based on the RankBoost algorithm. This algorithm, unlike most prior work, is tightly integrated with the search process, directing its learning based on the search performance of the current rule set. Our experiments show that this iterative learning approach is beneficial compared to a number of competitors in the context of greedy search.

The experiments show several domains where there is still significant room to improve. 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, Jiménez, and Borrajo 2008). However, we are also interested in understanding other failure modes of our approach and better understanding its convergence properties. 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.

Acknowledgments

This work was supported by NSF grant IIS-0546867.

	Learning iterations	Number of unique rules			Problems solved (Median plan length)		
		RB	RB-prior	ITR	RB	RB-prior	ITR
Blocksworld	5	3	4	5	**30(133)**	30(151)	30(160)
	10	7	8	10	30(126)	30(166)	**30(118)**
Depots	5	4	5	4	0(-)	2(8631)	**3(115)**
	10	7	9	8	3(9194)	4(954)	**20(796)**
	15	9	13	9	5(5372)	2(113)	**16(313)**
	20	11	17	12	2(5193)	7(263)	**23(433)**
	25	13	20	14	3(3188)	5(678)	**22(349)**
	30	16	24	15	15(661)	11(129)	**19(314)**
Driverlog	5	4	5	5	0(-)	1(1893)	**3(8932)**
	10	6	5	8	0(-)	**3(2852)**	1(2818)
	15	7	6	10	0(-)	0(-)	**3(544)**
	20	8	8	10	0(-)	1(4309)	**4(544)**
	25	9	9	10	0(-)	3(4082)	**4(544)**
	30	10	11	10	0(-)	1(632)	**3(544)**
FreeCell	5	2	4	5	2(213)	6(104)	**9(94)**
	10	4	7	5	2(186)	5(112)	**7(95)**
	15	7	10	6	3(103)	6(143)	**9(94)**
	20	11	15	6	5(155)	7(96)	**9(94)**
	25	11	19	6	3(334)	5(90)	**9(92)**
Pipesworld	5	3	4	3	4(382)	**17(1063)**	16(279)
	10	7	7	7	2(7845)	8(1821)	**11(307)**
	15	9	9	9	5(2929)	12(1599)	**17(595)**
	20	9	12	13	3(1369)	11(1423)	**17(579)**
	25	11	16	15	4(998)	11(2561)	**17(595)**
	30	12	19	18	7(1360)	12(1423)	**15(366)**
Pipesworld-with-tankage	5	5	5	4	0(-)	3(296)	**4(126)**
	10	6	8	9	0(-)	3(100)	**4(148)**
	15	9	9	10	0(-)	**4(98)**	4(134)
	20	12	12	10	1(1383)	**6(152)**	5(350)
	25	16	17	10	1(1383)	4(449)	**5(206)**
Philosopher	5	3	3	3	0(-)	**33(363)**	**33(363)**
	10	3	3	4	0(-)	**33(363)**	0(-)
	15	4	5	4	0(-)	**33(363)**	0(-)
	20	5	5	4	33(875)	**33(363)**	0(-)

Figure 4: Experimental results for different planners and different weighted rule sets. For each domain, we show the number of unique rules that are learned after the corresponding number of learning iterations. The performance of each learned rule set is given by the number of solved problems, together with 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.

References

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

Daumé III, H.; Langford, J.; and Marcu, D. 2009. Search-based structured prediction. *Machine Learning* 75:297–325.

de la Rosa, T.; Jiménez, S.; and Borrajo, D. 2008. Learning relational decision trees for guiding heuristic planning. In *ICAPS*, 60–67.

Freund, Y.; Iyer, R.; Schapire, R. E.; and Singer, Y. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* 4:933–969.

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

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

Kolobov, A.; Mausam, M.; and Weld, D. 2009. ReTrASE: integrating paradigms for approximate probabilistic planning. In *IJCAI*, 1746–1753. Morgan Kaufmann Publishers Inc.

Martin, M., and Geffner, H. 2000. Learning generalized policies in planning domains using concept languages. In *International Conference on Knowledge Representation and Reasoning*.

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

Xu, Y.; Fern, A.; and Yoon, S. 2009. Learning linear ranking functions for beam search with application to planning. *Journal of Machine Learning Research* 10:1571–1610.

Yoon, S.; Fern, A.; Givan, R.; and Kambhampati, S. 2008. Probabilistic planning via determinization in hindsight. In *AAAI*.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive policy selection for first-order MDPs. In *In Proceedings of Eighteenth Conference in Uncertainty in Artificial Intelligence*.

Yoon, S.; Fern, A.; and Givan, R. 2008. Learning control knowledge for forward search planning. *Journal of Machine Learning Research* 9:683–718.